ANL/NSE-19/18 Rev. 1

SAM User’s Guide

Nuclear Science & Engineering Division

ANL/NSE-19/18 Rev. 1

SAM User’s Guide

prepared by
Rui Hu, Ling Zou, Guojun Hu, Travis Mui
Nuclear Science & Engineering Division, Argonne National Laboratory

February 2021

ABSTRACT

The System Analysis Module (SAM) is a modern system analysis tool being developed at Argonne National Laboratory for advanced non-LWR safety analysis. It aims to provide fast-running, whole-plant transient analyses capability with improved-fidelity for Sodium-cooled Fast Reactors (SFR), Lead-cooled Fast Reactors (LFR), and Molten Salt Reactors (MSR) or Fluoride-cooled High-temperature Reactors (FHR). SAM takes advantage of advances in physical modeling, numerical methods, and software engineering to enhance its user experience and usability. It utilizes an object-oriented application framework (MOOSE), and its underlying meshing and finite-element library (libMesh) and linear and non-linear solvers (PETSc), to leverage the modern advanced software environments and numerical methods.

This document provides a user's guide, which will help users understand the input description and core capabilities of the SAM code. A brief overview of the code is presented, as well as how to obtain and run it. The input syntax for various parts of the code is provided. Additionally, a number of example problems, starting with simple unit component problems to problems with increasing complexity, are provided. Because the code is still under active development, this SAM User's Guide will evolve with periodic updates.

Contents

ABSTRACT ii

Contents vii

List of Figures viii

List of Tables ix

1 SAM Overview 1
- 1.1 Ultimate Goals and Objectives . . . 1
- 1.2 Software Structure . . . 2
- 1.3 Governing Theory . . . 2
  - 1.3.1 Fluid dynamics . . . 2
  - 1.3.2 Heat transfer . . . 3
  - 1.3.3 Closure models . . . 3
  - 1.3.4 Mass transport model development . . . 3
  - 1.3.5 Reactor kinetics model development . . . 4
  - 1.3.6 Numerical method . . . 4
- 1.4 Overview of Current Capabilities . . . 4

2 Running SAM 6
- 2.1 Pre-requisite . . . 6
- 2.2 Obtaining the Code . . . 6
- 2.3 Compiling the Code from Source . . . 7
- 2.4 Executing . . . 7
- 2.5 Outputs . . . 7

3 SAM Components 8

4 Input File Syntax 12
- 4.1 Global Parameters . . . 12
- 4.2 Equation of State (EOS) . . . 17
  - 4.2.1 PTEquationOfState . . . 17
  - 4.2.2 Built-in EOS . . . 18
  - 4.2.3 Simple Linearized EOS . . . 18
  - 4.2.4 PTFunctionsEOS . . . 19
  - 4.2.5 TabulatedEquationOfState . . . 21
  - 4.2.6 PTFluidPropertiesEOS . . . 22
- 4.3 Components . . . 23
  - 4.3.1 PBOneDFluidComponent . . . 23
  - 4.3.2 HeatStructure . . . 33
  - 4.3.3 PBPipe . . . 36
  - 4.3.4 PBCoreChannel . . . 39
  - 4.3.5 PBDuctedCoreChannel . . . 43

4.3.6 PBBypassChannel 45
4.3.7 PBMoltenSaltChannel 45
4.3.8 FuelAssembly 46
4.3.9 DuctedFuelAssembly 50
4.3.10 MultiChannelRodBundle 51
4.3.11 HexLatticeCore 54
4.3.12 PBCoupledHeatStructure 56
4.3.13 HeatStructureWithExternalFlow 59
4.3.14 HeatTransferWithExternalHeatStructure 61
4.3.15 PBHeatExchanger 62
4.3.16 PBTDJ 66
4.3.17 PBTDV 67
4.3.18 PressureOutlet 68
4.3.19 CoupledTDV 68
4.3.20 CoupledPPSTDJ 69
4.3.21 CoupledPPSTDV 69
4.3.22 PBSingleJunction 69
4.3.23 PBBranch 70
4.3.24 PBVolumeBranch 72
4.3.25 Valve 73
4.3.26 PBLiquidVolume 75
4.3.27 CoverGas 76
4.3.28 PBPump 77
4.3.29 StagnantVolume 78
4.3.30 LiquidTank 79
4.3.31 ReactorCore 80
4.3.32 SurfaceCoupling 81
4.3.33 ReactorPower 82
4.3.34 PointKinetics 83
4.3.35 ReferenceBoundary 87
4.3.36 PipeChain 88
4.3.37 ChannelCoupling 88
4.3.38 HeatPipe 89
4.3.39 MultiComponentArray 94
4.4 Control System Components 97
4.4.1 CTGeneric 97
4.4.2 ControlSystem 98
4.4.3 TripSystem 99
4.4.4 CSAddition 99
4.4.5 CSDivision 100
4.4.6 CSMultiplication 100
4.4.7 CSExponentiation 101
4.4.8 CSSTDFunction 102
4.4.9 CSDelay 102
4.4.10 CSUnitTrip 103

4.4.11 CSDifferentiation 104
4.4.12 CSIntegration 104
4.4.13 CSProportionIntegration 105
4.4.14 CSLeadLag 105
4.4.15 TSCompare 106
4.4.16 TSBoolean 107
4.4.17 TSDelay 107
4.4.18 Input syntax 108
4.5 ComponentInputParameters 110
4.6 PostProcessors 113
4.6.1 ComponentBoundaryEnergyBalance 113
4.6.2 ComponentBoundaryFlow 114
4.6.3 ComponentBoundaryScalarFlow 114
4.6.4 ComponentBoundaryVariableValue 114
4.6.5 ComponentNodalVariableValue 115
4.6.6 ConductionHeatRemovalRate 115
4.6.7 HeatExchangerHeatRemovalRate 116
4.7 TimeSteppers 116
4.7.1 CourantNumberTimeStepper 116
4.8 Preconditioning 117
4.9 Executioner 119
4.10 Outputs 125

5 Example Problems **134**
5.1 Heat Conduction Problem 134
5.2 Single Channel Flow 139
5.3 Core Channel 142
5.4 Heat Exchanger 146
5.5 Volume Branch 150
5.6 A Simple Loop Model 156
5.7 A Simplified SFR Model 161
5.8 Uncertainty Quantification using Dakota 179

6 Uncertainty Quantification **180**
6.1 Acquiring and Installing Dakota 180
6.2 Dakota and SAM Coupling 180
6.3 Multiple Node Execution (HPC cluster) 180
6.4 File structure overview 181
6.5 Example case: single node execution 183
6.5.1 Dakota Input File 183
6.5.2 Dakota-SAM Coupling File 184
6.5.3 Dakota Output 185
6.6 Example case: multiple node execution 185
6.6.1 Dakota Input File 185
6.6.2 Dakota-SAM Coupling File 186

6.6.3 Dakota Output 186

ACKNOWLEDGMENTS **187**

REFERENCES **187**

List of Figures

1.1 SAM Code Structure . . . 3
1.2 SAM simulation results of an SFR. . . . 5
4.1 SAM PBOneDFluidComponent examples. . . . 24
4.2 PBOneDFluidComponent with end element refinements. . . . 25
4.3 Fuel bundles in (a) square-lattice, typically seen in light water reactor designs; and (b) hexagonal-lattice, typically seen in sodium fast reactor designs. . . . 27
4.4 Typical SFR wire-wrapped rod configuration. . . . 28
4.5 An example of two-dimensional plate type of heat structure. . . . 34
4.6 SAM's PBPipe component, which consists of a `PBOneDFluidComponent` to model the one-dimensional fluid flow and one layer (or several layers) of `HeatStructure` to model its wall. . . . 36
4.7 An input example of PBPipe with two layers of heat structures to model its wall. For example, it could represent a layer of metal wall and an extra layer of thermal insulation material. . . . 37
4.8 PBCoreChannel component (a) SAM's PBCoreChannel component simulates the average coolant flow in rod bundles and heat conduction inside a fuel rod; and (b) An example mesh used in the PBCoreChannel component, 2-D mesh for heat structure and 1-D mesh for fluid flow. . . . 39
4.9 Sketch of the regions in the multi-channel model. . . . 51
4.10 Sketch of HexLatticeCore component. . . . 54
4.11 Two types of PBHeatExchanger component designs. As an example, the two figures show shell-and-tube heat exchanger design. . . . 63
4.12 The PBLiquidVolume concept used in SAM. (a) PBLiquidVolume with ambient pressure as its reference pressure; (b) PBLiquidVolume with an external CoverGas to specify its reference pressure. . . . 75
4.13 Sketch of a conventional 3-zone cylindrical heat pipe . . . 89
5.1 SAM model of the 2-D heat conduction problem. . . . 134
5.2 Comparisons of centerline temperature distributions of the heated rod, 2D conduction. 136
5.3 Example of SAM results shown in Paraview. . . . 139
5.4 Transient responses of the pipe under inlet temperature oscillation, BDF2. . . . 140
5.5 The schematic of the spatial discretization of the core channel problem. . . . 142
5.6 Temperature distribution of a counter-current heat exchanger. . . . 146
5.7 The three-pipe-in and two-pipe-out VolumeBranch test model. . . . 150
5.8 Input parameters of the three pipe in and two pipe out VolumeBranch test model. . 151
5.9 Transient temperature response at the VolumeBranch and pipe outlets. . . . 151
5.10 Schematics of the a test loop problem. . . . 156
5.11 Schematics of the a simple pool-type SFR model. . . . 161
6.1 Dakota-SAM Coupling Scheme . . . 181

List of Tables

2.1 Software Libraries Used by SAM . . . 6
3.1 List of boundary condition type of components of SAM . . . 8

3.2 List of junction type of components of SAM 9
3.3 List of non-geometric type of components of SAM 10
3.4 List of geometric and assembly type of components of SAM 11
4.1 SAM components that supports ComponentInputParameters feature 111
5.1 Comparison of SAM and Analytical Solutions for the Steady State Axial-Radial Heat Conduction Problem 135
6.1 Nominal Values of Uncertain Parameters 183

1 SAM Overview

The System Analysis Module (SAM) [1, 2, 3, 4] is an advanced system analysis tool being developed at Argonne National Laboratory under the U.S. Department of Energy (DOE) Nuclear Energy Advanced Modeling and Simulation (NEAMS) program. It aims to be a modern system analysis code, which takes advantage of the advancements software design, numerical methods, and physical models over the past two decades. SAM focuses on modeling advanced reactor concepts such as SFRs (sodium fast reactors), LFRs (lead-cooled fast reactors), and FHRs (fluoride-salt-cooled high temperature reactors) or MSRs (molten salt reactors). These advanced concepts are distinguished from light-water reactors (LWR) in their use of single-phase, low-pressure, high-temperature, and low Prandtl number (sodium and lead) coolants. This simple yet fundamental change has significant impacts on core and plant design, the types of materials used, component design and operation, fuel behavior, and the significance of the fundamental physics in play during transient plant simulations.

SAM is aimed to solve the tightly-coupled physical phenomena including heat generation, heat transfer, fluid dynamics, and thermal-mechanical response in reactor structures, systems and components in a fully-coupled fashion but with reduced-order modeling approaches to facilitate rapid turn-around for design and safety optimization studies. As a new code development, the initial effort focused on developing modeling and simulation capabilities of the heat transfer and single-phase fluid dynamics, as well as reactor point kinetics responses in reactor systems. This Chapter discusses goals and objectives, software structure, the governing theory, as well as current capabilities of the code. In the coming years, the SAM code will continuously mature as a modern system analysis tool for advanced (non-LWR) reactor design optimization, safety analyses, and licensing support.

1.1 Ultimate Goals and Objectives

The ultimate goal of SAM is to be used in advanced reactor safety analysis for design optimization and licensing support. The important physical phenomena and processes that may occur in reactor systems, structures, and components shall be of interest during reactor transients including Anticipated Operational Occurrence (AOO), Design Basis Accident (DBA), and additional postulated accidents but not including severe accidents. Typical reactor transients include: loss of coolant accidents, loss of flow events, excessive heat transfer events, loss of heat transfer events, reactivity and core power distribution events, increase in reactor coolant inventory events, and anticipated transients without scram (ATWS).

As a modern system analysis code, SAM is also envisioned to expand beyond the traditional system analysis code to enable multi-dimensional flow analysis, containment analysis, and source term analysis, either through reduced-order modeling in SAM or via coupling with other simulation tools. Additionally, the regulatory processes in the United States is being evolved to a risk-informed approach that is based on first understanding the best-estimate behavior of the fuel, the reactor, the reactor coolant system, the engineered safeguards, the balance of plant, operator actions, and all of the possible interactions among these elements. To enable this paradigm, an advanced system analysis code such as SAM must be able to model the integrated response of all of these physical systems and considerations to obtain a best-estimate simulation that includes both validation and uncertainty quantification.

The SAM code is aimed to provide improved-fidelity simulations of transients or accidents in an advanced non-LWR, including three-dimension resolutions as needed or desired. This will encompass the fuel rod, the fuel assembly, the reactor, the primary and intermediate heat transport system, the balance-of-plant, the containment. Multi-dimension, multi-scale, and multi-physics effects will be captured via coupling with other simulation tools, and computational accuracy and efficiency will be state-of-the-art. Uncertainty quantification will be integrated into SAM numerical simulations. Legacy issues such as numerical diffusion and stability in traditional system codes will be addressed and the code will attract broad use across the nuclear energy community based on its performance and many advantages relative to the legacy codes. The integrated architecture will provide a robust toolset for decision making with full consideration of the various disciplines and technologies affecting an issue.

1.2 Software Structure

SAM is being developed as a system-level modeling and simulation tool with higher fidelity (compared to existing system analysis tools), and with well-defined and validated simulation capabilities for advanced reactor systems. It provides fast-running, modest-fidelity, whole-plant transient analyses capabilities. To fulfill the code development, SAM utilizes the object-oriented application framework MOOSE [5] and its underlying meshing and finite-element library libMesh [6] and linear and non-linear solvers PETSc [7], to leverage the available advanced software environments and numerical methods. The high-order spatial discretization schemes, fully implicit and high-order time integration schemes, and the advanced solution method (such as the Jacobian-free Newton-Krylov (JFNK) method [8]) are the key aspects in developing an accurate and computationally efficient model in SAM.

The software structure of SAM is illustrated in Figure 1.1. In addition to the fundamental physics modeling of the single-phase fluid flow and heat transfer, SAM incorporates advances in the closure models (such as convective heat transfer correlations) for reactor system analysis developed over the past several decades. A set of Components, which integrate the associated physics modeling in the component, have been developed for friendly user interactions. This component-based modeling strategy is similar to what is implemented in RELAP-7 [9], which is also a MOOSE-based system analysis tool (focused on LWR simulations). A flexible coupling interface has been developed in SAM so that multi-scale, multi-physics modeling capabilities can be achieved by integrating with other higher-fidelity or conventional simulation tools.

1.3 Governing Theory

1.3.1 Fluid dynamics

Fluid dynamics is the main physical model of the SAM code. SAM employs a standard one-dimensional transient model for single-phase incompressible but thermally expandable flow. The governing equations consist of the continuity equation, momentum equation, and energy equations. A three-dimensional module is also under development to model the multi-dimensional flow and thermal stratification in the upper plenum or the cold pool of an SFR. Additionally, a subchannel module will be developed for fuel assembly modeling.

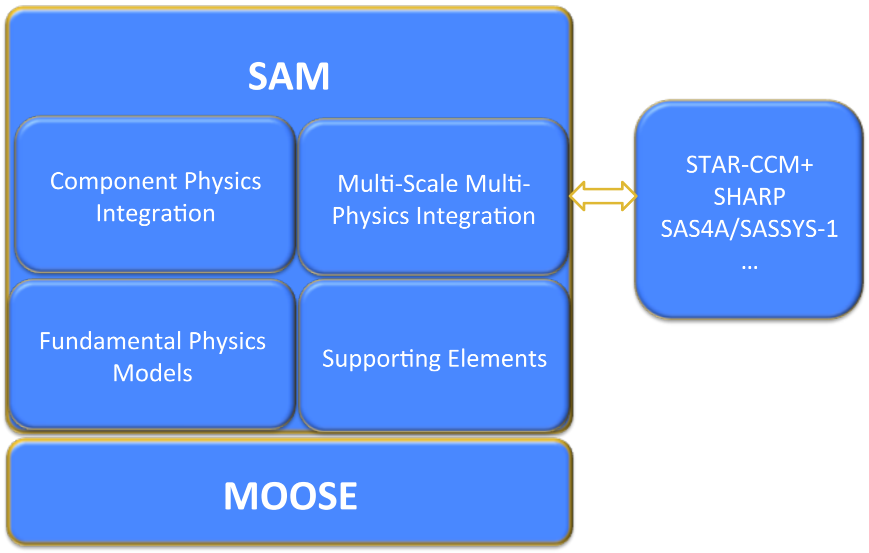

SAM
Component Physics Integration
Multi-Scale Multi-Physics Integration
Fundamental Physics Models
Supporting Elements
MOOSE
STAR-CCM+
SHARP
SAS4A/SASSYS-1
...

Figure 1.1: SAM Code Structure

1.3.2 Heat transfer

Heat structures model heat conduction inside solids and permit the modeling of heat transfer at interfaces between solid and fluid components. Heat structures are represented by one-dimensional or two-dimensional heat conduction in Cartesian or cylindrical coordinates. Temperature-dependent thermal conductivities and volumetric heat capacities can be provided in tabular or functional form. Heat structures can be used to simulate the temperature distributions in solid components such as fuel pins or plates, heat exchanger tubes, and pipe and vessel walls, as well as to calculate the heat flux conditions for fluid components. Flexible conjugate heat transfer and thermal radiation modeling capabilities are also implemented in SAM.

1.3.3 Closure models

The fluid equation of state (EOS) model is required to complete the governing flow equations, which are based on the primitive variable formulation; therefore, the dependency of fluid properties and their partial derivatives on the state variables (pressure and temperature) are implemented in the EOS model. Some fluid properties, such as sodium, air, salts like FLiBe and FLiNaK, have been implemented in SAM. Empirical correlations for friction factor and convective heat transfer coefficient are also required in SAM because of its one-dimension approximation of the flow field. The friction and heat transfer coefficients are dependent on flow geometries as well as operating conditions during the transient.

1.3.4 Mass transport model development

The mass transport modeling capability is needed to model sources and transport of particles for a number of applications, such as tritium transport, delayed neutron precursor drift, radioactive isotope transport for molten salt fueled/cooled systems. A general passive scalar transport model has been implemented in SAM, and it can be used to track any number of species carried by the fluid flow.

1.3.5 Reactor kinetics model development

SAM employs a built-in point kinetics model, including reactivity feedback and decay heat modeling. Various reactivity feedback mechanisms are included, such as the axial and radial expansion feedbacks due to thermal expansion and displacement effects. The effects of delay neutron precursor drift in MSRs can also be modeled.

1.3.6 Numerical method

SAM is a finite-element-method based code. The "weak forms" of the governing equations are implemented in SAM. It uses the Jacobian-Free Newton Krylov (JFNK) solution method to solve the equation system. The JFNK method uses a multi-level approach, with outer Newton's iterations (nonlinear solver) and inner Krylov subspace methods (linear solver), in solving large nonlinear systems. The concept of 'Jacobian-free' is proposed, because deriving and assembling large Jacobian matrices could be difficult and expensive. The JFNK method has become an increasingly popular option for solving large nonlinear equation systems and multi-physics problems, as observed in a number of different disciplines [8]. One feature of JFNK is that all the unknowns are solved simultaneously in a fully coupled fashion. This solution scheme avoids the errors from operator splitting and is especially suitable for conjugate heat transfer problems in which heat conduction in a solid is tightly coupled with fluid flow.

1.4 Overview of Current Capabilities

To develop a system analysis code, numerical methods, mesh management, equations of state, fluid properties, solid material properties, neutronics properties, pressure loss and heat transfer closure laws, and good user input/output interfaces are all indispensable. SAM leverages the MOOSE framework and its dependent libraries to provide JFNK solver schemes, mesh management, and I/O interfaces while focusing on new physics and component model development for advanced reactor systems. The developed physics and component models provide several major modeling features:

1. One-D pipe networks represent general fluid systems such as the reactor coolant loops.

2. Flexible integration of fluid and solid components, able to model complex and generic engineering system. A general liquid flow and solid structure interface model was developed for easier implementation of physics models in the components.

3. A pseudo three-dimensional capability by physically coupling the 1-D or 2-D components in a 3-D layout. For example, the 3-D full-core heat-transfer in an SFR reactor core can be modeled. The heat generated in the fuel rod of one fuel assembly can be transferred to the coolant in the core channel, the duct wall, the inter-assembly gap, and then the adjacent fuel assemblies.

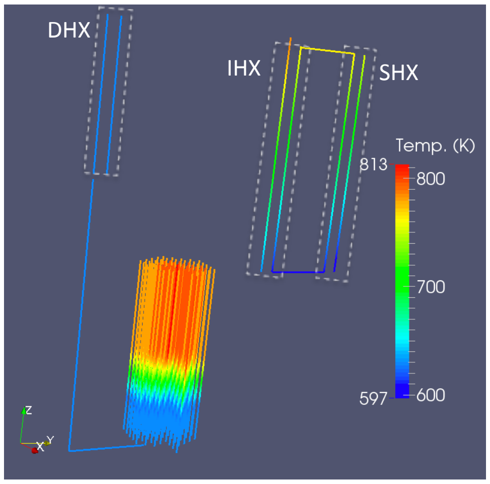
4. Pool-type reactor specific features such as liquid volume level tracking, cover gas dynamics, heat transfer between 0-D pools, fluid heat conduction, etc. These are important features for accurate safety analyses of SFRs or other advanced reactor concepts.

5. A computationally efficient multi-dimensional flow model is under development, mainly for thermal mixing and stratification phenomena in large enclosures for safety analysis. It was noted that an advanced and efficient thermal mixing and stratification modeling capability embedded in a system analysis code is very desirable to improve the accuracy of advanced reactor safety analyses and to reduce modeling uncertainties.

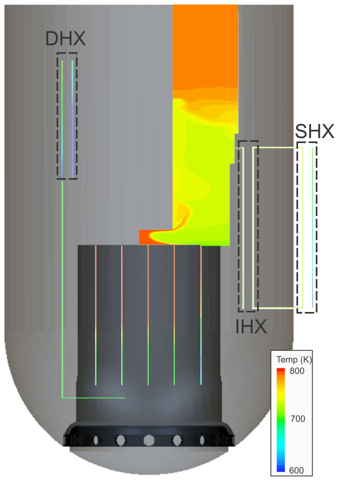
6. A general mass transport capability has been implemented in SAM based on the passive scalar transport. The code can track any number of species carried by the fluid flow for various applications.

7. An infrastructure for coupling with external codes has been developed and demonstrated. The code coupling with STAR-CCM+ [10], SAS4A/SASSYS-1 [11], Nek5000, and BISON [12] have been demonstrated, while the coupling with PRONGHORN, RattleSnake, and PORTEUS codes are ongoing or being planned.

An example of SAM simulation results of an SFR is shown in Figure 1.2.



(a) SAM model with 61 core channels



(b) Coupled SAM and CFD code simulation

Figure 1.2: SAM simulation results of an SFR.

2 Running SAM

2.1 Pre-requisite

SAM is built on the computational framework MOOSE (Multi-physics Object-Oriented Simulation Environment) to interface with LibMesh and PETSc to provide the underlying geometry (mesh I/O) and numerical capabilities (finite element library and solvers). It requires all of the code dependencies as MOOSE requires. A summary of the dependent libraries of SAM is listed in Table 2.1.

Table 2.1: Software Libraries Used by SAM

Library	Origin	Purpose
MOOSE [5]	Idaho National Laboratory	Computational framework, interfaces other libraries
LibMesh [6]	University of Texas, Austin	Finite element library
PETSc [7]	Argonne National Laboratory	Parallel linear and nonlinear solvers
Hypre (optional) [13]	Argonne National Laboratory	Parallel linear and nonlinear solvers
MPICH [14]	Argonne National Laboratory	Message passing/parallel processing
TBB (optional) [15]	Intel Corporation	Multi-thread parallelism

The MOOSE development team maintains a compiled set of all dependencies, except MOOSE and LibMesh, on the public MOOSE website (http://mooseframework.org/) with precompiled packages containing Petsc, Hypre, MPICH, and TBB for several Mac OS and Linux systems. MOOSE and LibMesh are available from the MOOSE GitHub site (https://github.com/idaholab/moose.git). For advanced users, all the dependent libraries are open- source codes and can thus be downloaded and compiled on Mac OS and Linux systems. The instructions for installing the MOOSE dependency package, and for compiling Libmesh and MOOSE can also be found at the public MOOSE website, http://www.mooseframework.org/.

2.2 Obtaining the Code

SAM is hosted in a private, access-controlled Git repository at Argonne National Laboratory. All changes to the source code are committed with revision number and comments, and are tracked in the repository. Contact the author of this User's Guide if interested in obtaining the code.

After obtaining access to the code, one could use git commands to obtain the source code of SAM:

```
git clone <repo_site_address>
```

MOOSE is set as a submodule of SAM, so that a reliable version of MOOSE is always available and consistent with the product version of SAM. The MOOSE submodule can be obtained by:

```
git submodule update -init
```

or by

```
git clone https://github.com/idaholab/moose.git moose
```

2.3 Compiling the Code from Source

After obtained the MOOSE submodule, one would need to compile libMesh first before compiling MOOSE and SAM. Under the moose/scripts directory, the libMesh can be obtained and compiled by:

```
./update_and_rebuild_libmesh.sh
```

After that, MOOSE and SAM can be compiled by use the default Makefile from the repository under the SAM folder. Use

```
make
```

to compile the code on a single processor; or use

```
make -j<n>
```

to compile the code on *n* processors.

2.4 Executing

SAM, due to its dependence on MOOSE, is not compatible with Windows operating systems. However it is fully compatible with Linux, Unix, and MacOS. It can be run from the shell prompt.

The execution command looks like:

```
sam-opt -i <input_file_name>
```

Many example test problems can be found under /tests/ subdirectory.

2.5 Outputs

SAM supports all MOOSE output file formats. It typically writes solution data to an ExodusII file, and write post-processor and scalar variables to a separate comma separated values (CSV) file. Several options exist for viewing ExodusII output files. One good choice is to use the open-source software Paraview (www.paraview.org). The CSV file uses table-structured format, which can be opened by many software such as Microsoft Excel.

3 SAM Components

The physics modeling (fluid flow and heat transfer) and mesh generation of individual reactor components are encapsulated as Component classes in SAM along with some component specific models. A set of components has been developed based on the finite-element fluid model and heat conduction model, including:

1. Basic geometric components;
2. 0-D components for setting boundary conditions;
3. 0-D components for connecting 1-D components;
4. Assembly components by combining the basic geometric components and the 0-D connecting components; and
5. Non-geometric components for physics integration.

A brief description of major SAM components is listed in Tables 3.1 - 3.4. The physics models associated with these components will be discussed in SAM Theory Manual, and the input format is discussed in Section 4.

Table 3.1: List of boundary condition type of components of SAM

Component name	**Descriptions**	**Dimension**
PBTDJ	An inlet boundary in which the flow velocity and temperature are provided by pre-defined functions.	0-D
CoupledPPSTDJ	CoupledPPSTDJ is a special PBTDJ component that is designed to facilitate MultiApp simulations.	0-D
PBTDV	A boundary in which pressure and temperature conditions are provided by pre-defined functions.	0-D
CoupledTDV	A time-dependent-volume boundary in which boundary conditions are provided by other codes in coupled code simulation.	0-D
CoupledPPSTDV	CoupledPPSTDV is a special PBTDV component that is designed to facilitate MultiApp simulations.	0-D
PressureOutlet	A subset of PBTDV, will be removed.	0-D
ReferenceBoundary	ReferenceBoundary component provides a fixed value boundary condition to a one-dimensional fluid type of component.	0-D
StagnantVolume	Models a stagnant liquid volume, with connections to other 0-D volumes but no connections to 1-D fluid components.	0-D

Table 3.2: List of junction type of components of SAM

Component name	**Descriptions**	**Dimension**
PBSingleJunction	Models a zero-volume flow joint, where only two 1-D fluid components are connected.	0-D
PBBranch	Models a zero-volume flow joint, where multiple 1-D fluid components are connected.	0-D
PBVolumeBranch	Considering the volume effects of a PBBranch component so that it can account for the mass and energy in-balance between inlets and outlets due to inertia.	0-D
PBPump	Simulates a pump component, in which the pump head is dependent on a pre-defined function.	0-D
PBLiquidVolume	A 0-D liquid volume with cover gas (the liquid level is tracked and the volume can change during the transient).	0-D
LiquidTank	The LiquidTank component of SAM simulates a PB-VolumeBranch (or PBLiquidVolume) and the heat structure (modeled as PBCoupledHeatStructure) attached to it in order to capture this additional thermal inertia.	0-D fluid, 1-D or 2-D structure

Table 3.3: List of non-geometric type of components of SAM

Component name	**Descriptions**	**Dimension**
ReactorCore	Models a pseudo three-dimensional reactor core; It consists of member core channels (with duct walls) and bypass channels.	1-D fluid, 1-D or 2-D structure
CoverGas	A 0-D gas volume that is connected to one or multiple liquid volumes.	0-D
SurfaceCoupling	The SurfaceCoupling component models the heat transfer between two solid surfaces, suitable for radiation heat transfer or gap heat transfer between them.	ND
ChannelCoupling	A non-geometric component for coupling two 1-D fluid ND components (with energy exchange).	ND
ReactorPower	A non-geometric component describing the total reactor ND power.	ND
PointKinetics	The PointKinetics component is the build-in point kinetics model of SAM, which models the transient behaviors of reactor fission power, delayed-neutron precursors, as well as reactivity feedback from other components, e.g., core channels.	ND
PipeChain	A non-geometric component for connecting a number of ND fluid components.	ND
HeatTransferWith ExternalHeatStructure	A non-geometric component for connecting a number of ND fluid components.	ND

Table 3.4: List of geometric and assembly type of components of SAM

Component name	**Descriptions**	**Dimension**
PBOneDFluidComponent	Simulates 1-D fluid flow using the primitive variable based fluid model	1-D
HeatStructure	Simulates 1-D or 2-D heat conduction inside solid structures	1-D or 2-D
PBCoupledHeatStructure	The heat structure connecting two liquid components (1-D or 0-D).	1-D or 2-D
PBPipe	Simulates fluid flow in a pipe and heat conduction in the pipe wall.	1-D fluid, 1-D or 2-D structure
PBHeatExchanger	Simulates a heat exchanger, including the fluid flow in the primary and secondary sides, convective heat transfer, and heat conduction in the tube wall.	1-D fluid, 1-D or 2-D structure
PBCoreChannel	Simulates reactor core channels, including 1-D flow channel and the inner heat structures (fuel, gap, and clad) of the fuel rod.	1-D fluid, 1-D or 2-D structure
PBDuctedCoreChannel	Simulates reactor core channels with an outer heat structure of the duct wall.	1-D fluid, 1-D or 2-D structure
PBBypassChannel	Models the bypass flow in the gaps between fuel assemblies.	1-D
FuelAssembly	Models reactor fuel assemblies composed of multiple CoreChannels, representing different regions of a fuel assembly (core, gas plenum, reflector, shield, etc.).	1-D fluid, 1-D or 2-D structure
DuctedFuelAssembly	Model reactor fuel assemblies composed of multiple DuctedCoreChannels.	1-D fluid, 1-D or 2-D structure
MultiChannelRodBundle	Models the rod bundle with a multi-channel model, in which multiple CoreChannels and the inter-channel mixing are defined and created.	1-D fluid, 1-D or 2-D structure
HexLatticeCore	Models a hexagonal lattice core, in which the CoreChannels and HeatStructures are defined and created.	1-D fluid, 1-D or 2-D structure
PBMoltenSaltChannel	PBMoltenSaltChannel is a component intended to model the core behavior of molten-salt reactor designs.	1-D
HeatPipe	HeatPipe is a component to model heat pipes.	1-D fluid, 1-D or 2-D structure
HeatStructure WithExternalFlow	HeatStructureWithExternalFlow is also a HeatStructure-based component similar to PBCoupledHeatStructure, however with the main purpose to facilitate code-to-code coupling via its boundary surfaces.	1-D or 2-D structure

4 Input File Syntax

SAM uses a block-structured input file. Each block is identified with square brackets. The opening brackets contain the type of the input block and the empty brackets mark the end of the block. Each block may contain sub-blocks. Each sub-block must have a unique name when compared with all other sub-blocks in the current block.

Line inputs are given as parameter and value pairs with an equal sign between them. They specify parameters to be used by the object being described. The parameter is a string, and the value may be a string, a Boolean value, an integer, a real number, or a list of strings, integers, or real numbers. Lists are given in single quotes and are separated by whitespace. Sub-blocks normally contain a type line input. This line specifies the particular type of object being described.

All units used in SAM are SI units. This standardizes the model input by eliminating the possibility of errors caused by using one set of units for one model and another set of units for a different model. "#" symbol indicates comments in the input file and can be located anywhere in the input file.

A quick example is given to demonstrate the basic block-structured syntax of SAM input file:

```
[BlockName]                                 # Beginning of an input block
  RealNumber          = 1.0                 # This specifies a real number
  Boolean             = true                # This specifies a boolean value
  MyString            = SAM                 # This specifies a string value
                                            # An empty line will be simply ignored
  [./SubBlockName]                          # Beginning of an input sub-block
    Numbers           = '1.0  2.0  3.0'     # This specifies a list of numbers
    Strings           = 'Hello  World'      # This specifies a list of strings
  [../]                                     # Ending of an input sub-block
[]                                          # Ending of an input block
```

The following subsections have brief descriptions of each block used in SAM input. This User's Guide is intended to help users understand the basics of the SAM code and learn how to run it. The details of the input parameters and modeling options will be discussed in a more detailed User's Manual in the future when the SAM code becomes more mature.

4.1 Global Parameters

The GlobalParams block specifies the global parameters used by the code such as global initial conditions, the scaling factors for the primary variable residuals, etc. The modeling parameters associated with the primitive-variable-based fluid model can be defined in the PBModelParams sub-block.

The full list of input parameters of the GlobalParams block is shown below. The line inputs are listed in a three-column format, with the first column showing the available input parameters, the second column showing the default value of the input parameters, and the third column showing a short description of the input parameters. "= (required)" is listed in some cases in the second column for parameter in other input blocks, which indicates that the parameter must be provided in the input file otherwise the code cannot be executed.

```
[GlobalParams]
  SC_HTC              = 1               # Sensitivity coefficient for HTC
  SC_WF               = 1               # Sensitivity coefficient for wall friction
  Tsolid_sf           = 0.001           # Scaling factor for solid temperature variable.
  active              = __all__         # If specified only the blocks named will be
                                        # visited and made active
  global_init_P      = 100000          # Global initial fluid pressure
  global_init_T      = 300             # Global initial temperature for fluid and solid
  global_init_V      = 0.0001          # Global initial fluid velocity
  gravity            = '0 0 -9.8'      # Gravity vector
  inactive           = (no_default)    # If specified blocks matching these identifiers
                                       # will be skipped.
  model_type         = 1               # Which physical model to use (currently not
                                       # in use)
  scaling_factor_var = '1 0.001 1e-06' # Scaling factors for fluid variables (p, v,
                                       # T)

  [./PBModelParams]
    Courant_control               = (no_default) # If to set the dt according to the
                                                 # target Courant number
    P_bounds                      = '0 1e+08'    # Lower and upper bounds for pressure
                                                 # variable
    T_bounds                      = '100 1200'   # Lower and upper bounds for temperature
                                                 # variable
    V_bounds                      = '-1000 1000' # Lower and upper bounds for velocity
                                                 # variable
    active                        = __all__      # If specified only the blocks named
                                                 # will be visited and made active
    decay_heat_precursor          = (no_default) # The name of decay heat precursor
                                                 # in fluid transport
    fluid_conduction              = 0            # If modeling axial fluid  conduction
    global_init_PS                = (no_default) # The global initial value of passive
                                                 # scalar in fluid transport
    inactive                      = (no_default) # If specified blocks matching these
                                                 # identifiers will be skipped.
    low_advection_limit           = 1e-07        # Lower bound of velocity for advection
                                                 # dominant region
    p_order                       = 1            # P-order of the mesh
    passive_scalar                = (no_default) # The name of passive scalar in fluid
                                                 # transport
    passive_scalar_decay_constant = (no_default) # The decay constant of passive scalar
                                                 # (e.g. delayed neutron precursors
                                                 # or decay heat precursors in fluid
                                                 # transport
    passive_scalar_diffusivity    = (no_default) # The diffusivity of passive scalar
                                                 # in fluid transport
    pbm_scaling_factors           = (no_default) # Scaling factors for each variable
    pspg                          = 1            # If using pspg stabilization scheme
    scaling_velocity              = (no_default) # Global scaling velocity for PSPG
    supg                          = 1            # If using supg stabilization scheme
    supg_max                      = 0            # If using pspg stabilization scheme
    variable_bounding             = (no_default) # If using variable bounding
    total_mass_check              = 0            # If turning on built-in total mass pps
    mass_flow_rate_check          = 0            # If truning on built-in mass flow rate pps
    net_mass_flow_rate_check      = 0            # If turning on built-in net mass flow rate pps
    energy_balance_check          = 0            # If turning on built-in energy balance pps
  [../]
[]
```

For each input parameter in the `GlobalParams` input block, details are provided as follows:

- `global_init_P`

  As the name suggests, it specifies the global initial value of fluid pressure, which however can be overridden by initial values specified locally in the component level, for example, initial_P of PBOneDFluidComponent (section 4.3.1).

  If not specified, a default value, 10^5 Pa, is used for this parameter.

- `global_init_V`

  This input parameter specifies the global initial value of fluid velocity, which can also be overridden by initial values specified locally in the component level, for example, initial_V of PBOneDFluidComponent (section 4.3.1).

  If not specified, a default value, 10^{-4} m/s, is used.

- `global_init_T`

  This input parameter specifies the global initial value of fluid temperature, which can also be overridden by initial values specified locally in the component level, for example, initial_T of PBOneDFluidComponent (section 4.3.1).

  If not specified, a default value, 300 K, is used.

- `scaling_factor_var`

  This input parameter specifies the scaling factors to the residuals of three fluid equations, i.e., mass, momentum, and energy equations. The default values are '`1 0.001 1e-06`', which general work pretty well for most cases.

- `Tsolid_sf`

  This input parameter specifies the scaling factor to the residual of heat conduction equation in solids, e.g., heat structures. The default value is 0.001.

- `SC_HTC`

  This input parameter works as a global multiplier to the heat transfer coefficient used in the code, e.g., $HTC_{new} = HTC_{original} \times SC\_HTC$. Similar to those global initial values, this parameter can also be overridden locally in the component level, for example, SC_HTC of PBOneDFluidComponent (section 4.3.1).

- `SC_WF`

  This input parameter works as a global multiplier to the wall friction coefficient used in the code, e.g., $f_{new} = f_{original} \times SC\_WF$. This parameter can also be overridden locally in the component level, for example, SC_WF of PBOneDFluidComponent (section 4.3.1).

- `gravity`

  This input parameter is the vector form of gravitational constant in (x,y,z) coordinates. The default value is '`0 0 -9.8`'.

- `model_type`

  This input parameter specifies the model type used in SAM simulation, 1 for one-dimensional model (default value), and 2 for three-dimensional model. However, currently this input parameter is not used.

Input parameters of the input sub-block, `PBModelParams`, is discussed as follows:

- `Courant_control`

  This is a boolean type of input parameter, by default, false. If specified true, the code uses the maximum Courant number (automatically calculated) as an indicator to control the time step size during a transient simulation. It is used in combination of the `CourantNumberTimeStepper` PostProcessor, see section 4.7.1.

- `variable_bounding`

  This input parameter specifies if variables bounding should be applied to the main fluid variables, i.e., pressure, velocity, and temperature. By default, it is false, i.e., no bounding is applied.

- `P_bounds`, `T_bounds`, and `V_bounds`

  These input parameters specify the bounds for the three main fluid variables. The default values are: `P_bounds = '0 1.0e8'` Pa, `V_bounds = '-1.0e-3 1.0e3'` m/s, and `T_bounds = '100 1.2e3'` K. These bounds are only applied when `variable_bounding = true`.

- `fluid_conduction`

  This input parameter specifies if axial heat conduction effect of the fluid should be modeled, which, if modeled, would be included in the fluid energy equation. Such an effect is generally only important in applications where high thermal-conductivity fluids, such as liquid metals, are used. An example application is sodium-cooled fast reactor analysis. For most other applications, it is safe to not include this effect.

- `passive_scalar`

  This input parameter accepts a list of names of passive scalars that are passively transported with fluid flow. For example, `passive_scalar = 'particle1 particle2 particle3'`.

- `global_init_PS`

  This input parameter specifies the global initial values of passive scalars. For example, `global_init_PS = '10.0 80.0 20.0'`. Similar to fluid properties, such as pressure, this global initial condition could be overridden by locally specified initial conditions in in the component level, for example, initial_PS of PBOneDFluidComponent (section 4.3.1).

- `passive_scalar_diffusivity`

  This input parameter specifies the diffusivities of passive scalars in fluid.

- `passive_scalar_decay_constant`

  This input parameter specifies the decay constants of passive scalars. If part of the `passive_scalar` list is also defined as `decay_heat_precursor`, the corresponding decay constants will be used as decay heat precursor decay constant to compute decay power.

- `decay_heat_precursor`

  This input parameter defines a list of decay heat precursors, each of which must have been specified in the `passive_scalar` list, to compute decay power.

- `p_order`

  This input parameter specifies the p-order of one- and two-dimensional meshes generated within the code. The default value is 1, i.e., first-order.

- `pbm_scaling_factors`

  This input parameter works similarly to the higher level global input parameter, `scaling_factor_var`. If specified, it overrides `scaling_factor_var`.

- `pspg`

  This input parameter specifies if PSPG stabilization should be used in the fluid mass equation. By default, it is true (1).

- `scaling_velocity`

  This input parameter specifies a reference velocity for scaling to be used in the PSPG scheme. Currently, not used.

- `supg`

  This input parameter specifies if SUPG stabilization should be used in the fluid momentum and energy equations. By default, it is true (1).

- `supg_max`

  In some extreme cases, for example, fluid velocities very close to 0. The FEM scheme may not be stable enough to cause unphysical oscillations in numerical solutions. With `supg_max = true`, stabilization parameters are adjusted to larger values that help suppress such non-physical oscillations. In most cases, this is not needed, and it is false (0), by default.

- `low_advection_limit`

  This parameter specifies the lower bound of velocity for advection dominant region. When the velocity magnitude is smaller than this value, SUPG stabilization scheme is deemed to be unnecessary, and is turned off. The default value of this input parameter is 10^{-7} m/s.

- `total_mass_check`

  This parameter specifies if turning on the built-in total mass postprocessor for checking the mass in different 1-D fluid components, 0-D volumes, and fluid loops.

- `mass_flow_rate_check`

  This parameter specifies if turning on the built-in mass flow rate postprocessor for obtaining the mass flow rate in 1-D fluid components.

- `net_mass_flow_rate_check`

  This parameter specifies if turning on the built-in net mass flow rate postprocessor for obtaining the mass flow rate in 1-D fluid components.

- `energy_balance_check` This parameter specifies if turning on the built-in energy balance postprocessor for obtaining the energy balance rate in 1-D fluid components.

An example input of the GlobalParams block is shown below. Note that only a small fraction of the parameters were provides. For other unprovided input parameters, default values are used if they are available in the code (as listed in the above input description). If the default value is not available, the parameter is not required and its intended function is not activated.

```
[GlobalParams]
  global_init_P = 1.2e5
  global_init_V = 1
  global_init_T = 628.15
  scaling_factor_var = '1 1e-3 1e-6'
  [./PBModelParams]
    p_order = 2
  [../]
[]
```

Another example is given on passive scalars. There are eight passive scalars specified in `PBModelParams`, six of which are also defined as `decay_heat_precursor`.

```
[GlobalParams]
  global_init_P = 1.1e5
  global_init_V = 0.5
  global_init_T = 628.15
  Tsolid_sf = 1e-1

  [./PBModelParams]
    pbm_scaling_factors = '1 1e-3 1e-6'
    passive_scalar = 'TEST235-group0 TEST235-group1 TEST235-group2 TEST235-group3
                      TEST235-group4 TEST235-group5 c1 c6'
    passive_scalar_diffusivity = '0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0'
    passive_scalar_decay_constant = '2.722  1.026  0.314  0.118  0.034  0.012  0.0124
                                     3.010'
    global_init_PS = '0.0  0.0  0.0  0.0  0.0  0.0  1.94944E+01 3.45288E-13'
    p_order = 2
    Courant_control = true
    decay_heat_precursor = 'TEST235-group0 TEST235-group1 TEST235-group2 TEST235-group3
                            TEST235-group4 TEST235-group5'
  [../]
[]
```

4.2 Equation of State (EOS)

SAM provides different options in specifying fluid properties in simulations. Users could choose from SAM's built-in fluid library for commonly-used fluids, including air, nitrogen, helium, sodium, two types of molten salt (Flibe and Flinak), and one simulant oil (DowthermA).

4.2.1 PTEquationOfState

`PTEquationOfState` is an abstract base class for the various EOS provided by SAM, thus it cannot be directly used in an input model. It takes several input parameters that are common for specific EOS given in the following subsections. These parameters are listed and explained below.

```
[./PTEquationOfState]
  type           = PTEquationOfState
  p_0            = 100000   # Reference pressure
  eos_test       = false    # Option to output fluid properties to a CSV file
  eos_test_Tmin  = 273.15   # Minumum temperature for output
  eos_test_Tmax  = 2273.15  # Maximum temperature for output
  eos_test_num   = 201      # Number of temperature nodes for output
```

```
[../]
```

- p_0

  A reference pressure with default value of 10^5 Pa. For this EOS, it is not used and safe to leave it unspecified.

- eos_test, eos_test_Tmin, eos_test_Tmax, eos_test_num

  With this set of input parameters, the user can output all fluid properties as a function of temperature to a CSV file. This feature allows users checking the consistency of provided EOS in a test run. The output file is named by: 'eosName-SAMTest.csv'. Tips: users are recommended to double-check that there are no non-physical values in fluid properties because it brings in convergence difficulty to the code.

4.2.2 Built-in EOS

SAM provides several built-in EOS for users to pick from. These model requires minimum input effort, and examples are given as follows:

```
[./air_eos]
  type = AirEquationOfState
[../]
[./Helium]
  type = HeEquationOfState
[../]
[./N2]
  type = N2EquationOfState
[../]
[./sodium]
  type = PBSodiumEquationOfState
[../]
[./eos]
  type = SaltEquationOfState
  salt_type = Flibe
[../]
[./eos]
  type = SaltEquationOfState
  salt_type = Flinak
[../]
[./eos]
  type = SaltEquationOfState
  salt_type = DowthermA
[../]
```

4.2.3 Simple Linearized EOS

SAM also provides another simple equation of state, in which all properties, except density and specific enthalpy, are constant user-specified input values. The complete input parameters of this simple equation of state is given as follows:

```
[./PTConstantEOS]
  SC_cp    = 1             # Sensitivity coefficient for heat capacity
  SC_k     = 1             # Sensitivity coefficient for thermal conductivity
  SC_mu    = 1             # Sensitivity coefficient for viscosity
  SC_rho   = 1             # Sensitivity coefficient for density
  T_0      = (required)    # Reference temperature
  beta     = 0             # Coefficient of thermal expansion
  cp       = (required)    # Specific heat
  cv       = (no_default)  # Specific heat
  h_0      = (required)    # Reference internal enthalpy
  k        = (required)    # Thermal conductivity,W/(m-K)
  mu       = (required)    # Dynamic viscosity, Pa.s
  p_0      = 100000        # Reference pressure
  rho_0    = (required)    # Reference density
  type     = PTConstantEOS
[../]
```

Density is a linear function of temperature using the provided thermal expansion coefficient, β, which is calculated as:

$$\rho = \rho_0 - \rho_0\beta(T - T_0)$$

Tips: a non-zero thermal expansion coefficient is needed to provide the driven force for natural circulation flow in a close loop. Specific enthalpy is also linearly dependent on temperature,

$$h = h_0 + c_p(T - T_0)$$

- `SC_cp, SC_k, SC_mu, SC_rho`

  These are sensitivity coefficients that are multiplied to the values of specific heat, thermal conductivity, viscosity, and density of the fluid. They are most useful for uncertainty quantification, and by default, are zero. For normal applications, they could be simply ignored. These parameters are available for all equation of states implemented in SAM code, including those built-in fluid library discussed earlier.

- `p_0`

  A reference pressure with default value of 10^5 Pa. For this EOS, it is not used and safe to leave it unspecified.

Other input parameters are self-explanatory and thus not discussed further. An example is given as follows:

```
[EOS]
  [./eos]
    type = PTConstantEOS
    p_0 = 1e5          # Pa
    rho_0 = 865.51     # kg/m^3
    beta = 2.7524e-4   # K^{-1}
    cp =  1272.0       # J/kg-K, at Tavg;
    h_0 = 7.9898e5     # J/kg
    T_0 = 628.15       # K
    mu = 2.6216e-4     # Pa-s
    k = 72             # W/m-K
  [../]
[]
```

4.2.4 PTFunctionsEOS

In addition to the simple linearized equation of state, SAM also provides PTFunctionsEOS to accept more complex user-defined fluid properties in terms of pressure and temperature-dependent functions. Its input parameters are listed as follows:

```
[./PTFunctionsEOS]
  type       = PTFunctionsEOS
  p_0        = 100000     # Reference pressure
  SC_cp      = 1          # Sensitivity coefficient for heat capacity
  SC_k       = 1          # Sensitivity coefficient for thermal conductivity
  SC_mu      = 1          # Sensitivity coefficient for viscosity
  SC_rho     = 1          # Sensitivity coefficient for density
  cp         = (required) # Specific heat capacity
  rho        = (required) # Density
  k          = (required) # Thermal conductivity, W/(m-K)
  mu         = (required) # Dynamic viscosity, Pa.s
  enthalpy   = (optional) # Specific enthalpy
  T_min      = (optional) # Valid minimum temperature for built-in enthalpy calculation
  T_max      = (optional) # Valid maximum temperature for built-in enthalpy calculation
  T_nodes    = 51         # Number of temperature nodes for built-in enthalpy calculation
  h_0        = 0          # Reference specific enthalpy at the minimum valid temperature
[../]
```

Among these input parameters, `SC_cp`, `SC_k`, `SC_mu`, `SC_rho`, and `p_0`, are the same as described in section 4.2.3. Other parameters are described as follows:

- `rho`, `cp`, `mu`, `k` (required)

  All these input parameters are required. Each of them accepts either a constant value or a function name, which should have been specified in the `[Functions]` input block.

- `enthalpy`

  The input parameter `enthalpy` specifies the user-provided function name for specific enthalpy. The function itself should have been defined in `[Functions]` input block. Warning: user needs to ensure the consistency between specific enthalpy and specific heat capacity, i.e. specific enthalpy is integrated from the specific capacity; otherwise, convergence difficulty could show up.

- `T_min`, `T_max`, `T_nodes`, and `h_0`

  The users are advised to ignore the `enthalpy` input and provide a valid temperature range for built-in specific enthalpy calculation. Given the valid temperature range with input parameters `T_min`, `T_max`, and `T_nodes`, the code will calculate the specific enthalpy based on the user-provided specific heat capacity function. The built-in calculation is performed by:

$$h(T) = h_i + \frac{C_p(T) + C_p(T_i)}{2}(T - T_i) \tag{4.1}$$

$$T_i = T_{min} + i \times \frac{T_{max} - T_{min}}{T_{nodes} - 1}, \text{ for } i = 0, 1, \cdots, T_{nodes} - 1 \tag{4.2}$$

  where T_i is the lower-bound temperature node for T and h_i is the specific enthalpy pre-calcuated with this formulation on the temperature node T_i. `h_0` provides the reference specific enthalpy at the minimum valid temperature `T_min`. The default value of 0 is a good choice for the reference specific enthalpy.

An example of using ‘PTFunctionsEOS’ is given as follows:

```
[Functions]
  [./enthalpy_fn]
```

```
    type = PiecewiseLinear
    x = '428.15        628.15      1028.15' # 'x' really means temperature.
    y = '5.4458e5    7.9898e5    1.30778e6'
  [../]
[]

[EOS]
  [./eos]
    type = PTFunctionsEOS
    rho  = 865.51
    cp   = 1272.0
    mu   = 2.6216e-4
    k    = 72
    enthalpy = enthalpy_fn
  [../]
[]
```

4.2.5 TabulatedEquationOfState

It is often the case users know fluid properties at discrete temperature nodes instead of a continuous function of temperature, TabulatedEquationOfState is an additional equation of state to accept user-provided temperature-dependent fluid properties. Its input parameter list is given as follows:

```
[./TabulatedEquationOfState]
  type        = TabulatedEquationOfState
  SC_cp       = 1             # Sensitivity coefficient for heat capacity
  SC_k        = 1             # Sensitivity coefficient for thermal conductivity
  SC_mu       = 1             # Sensitivity coefficient for viscosity
  SC_rho      = 1             # Sensitivity coefficient for density
  temperature = (required)    # Vector of temperature
  rho         = (required)    # Vector of density
  cp          = (required)    # Vector of specific heat capacity
  mu          = (required)    # Vector of viscosity
  k           = (required)    # Vector of thermal conductivity
  interp_type = Linear        # Type of interpolation: Linear or Spline
  h_0         = 0             # Reference specific enthalpy at minimum valid temperature
[../]
```

Among these input parameters, SC_cp, SC_k, SC_mu, and SC_rho are the same as described in section 4.2.3. Other parameters are described as follows:

- temperature (required)

  This is a required parameter that accepts the vector temperature listed in an increasing order. The valid temperature range for this EOS is the first and last entry in this vector.

- rho, cp, mu, k (required)

  These are requires parameters that accepts the vector of density, specific heat capacity, viscosity, and thermal conductivity listed corresponding to the temperature entry specified for temperature. The vector size of temperature, rho, cp, mu, and k should be equal.

- interp_type

  This parameter specifies how the fluid properties are interpolated based on temperature. The options are ‘Linear’ and ‘Spline’. Linear and cubic spline interpolation will be used for ‘Linear’ and ‘Spline’, respectively.

- h_0

  This parameter specifies the reference specific enthalpy at the minimum valid temperature. If this reference value is not known, the default value of 0 is recommended.

An example of using ‘TabulatedEquationOfState’ is given as follows:

```
[EOS]
  [./eos]
    type        = TabulatedEquationOfState
    interp_type = Linear
    temperature = '275       280       285       290       295'
    rho         = '999.94    999.91    999.52    998.80    997.81'
    cp          = '4213.34   4200.97   4192.80   4187.31   4183.59'
    k           = '0.5659    0.5760    0.5854   0.5942    0.6025'
    mu          = '1.682E-03 1.434E-03 1.239E-03 1.084E-03 9.578E-04'
    h_0         = 7859.83
  [../]
[]
```

4.2.6 PTFluidPropertiesEOS

To take advantages of many existing built-in fluid properties provided within the MOOSE framework, SAM provides an “interface” class, PTFluidPropertiesEOS, to access these fluid property libraries. Its input parameter list is given as follows:

```
[./PTFunctionsEOS]
  SC_cp     = 1                # Sensitivity coefficient for heat capacity
  SC_k      = 1                # Sensitivity coefficient for thermal conductivity
  SC_mu     = 1                # Sensitivity coefficient for viscosity
  SC_rho    = 1                # Sensitivity coefficient for density
  fp        = (required)       # The name of the user object for fluid properties
  type      = PTFluidPropertiesEOS
[../]
```

Other than the four sensitivity coefficients, the only user input is a name pointing to a MOOSE-provided fluid property library. This is a required input parameter:

- fp (required)

  This is a required parameter that accepts the name of the user object for a MOOSE-provided fluid library. This user object should have been provided in a separate material properties input block, [MaterialProperties].

An example of PTFluidPropertiesEOS usage is given as:

```
[EOS]
  [./eos]
    type = PTFluidPropertiesEOS
    fp = fluid_props                # Pointing to a user object provided
                                    # in the following MaterialProperties block
  [../]
[]

[MaterialProperties]
  [./fluid_props]
    type = IdealGasFluidProperties  # MOOSE-provided fluid library
    gamma = 1.4
    R = 286.9
    mu = 2.e-5 #Pa-s
    k =  0.03
  [../]
[]
```

4.3 Components

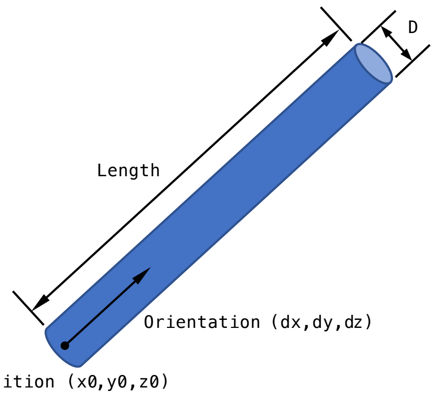
4.3.1 PBOneDFluidComponent

`PBOneDFluidComponent` is the most basic fluid component in SAM. It represents a unit one-dimensional (1D) component to simulate the 1D fluid flow in a channel. The geometry parameters such as the hydraulic diameter, flow area, and length, are provided in the input file. The wall friction and heat transfer coefficients can be calculated through the closure models based on flow conditions and geometries or provided by the user input. Internal volumetric heating (or cooling) can be specified by the user input as well. The associated input parameters of the PBOneDFluidComponent Component block are shown below.

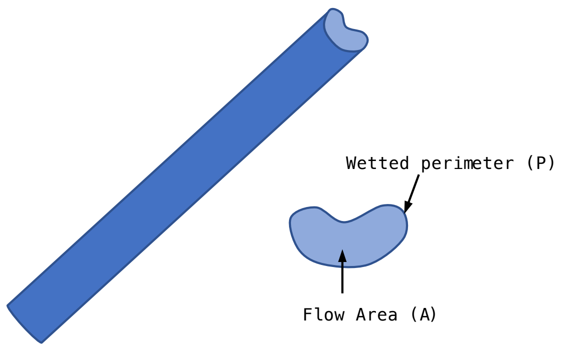
```
[./PBOneDFluidComponent]
  A                        = (required)       # Area of the One-D fluid component
  Cgb                      = 1                # Mixing coefficient due to buoyancy
                                              # and geometry effects
  Cgv                      = (no_default)     # Mixing coefficient due to velocity
                                              # and geometry effects
  Dh                       = (required)       # Hydraulic diameter
  HTC_geometry_type        = Pipe             # Heat transfer geometry type
  HTC_user_option          = Default          # Heat transfer correlation user option
  HT_surface_area_density  = (no_default)     # Heating surface density
  HoD                      = 1                # wire pitch ratio, height to diameter
  Hw                       = (no_default)     # Convective heat transfer coefficient
  Ph                       = (no_default)     # Heated perimeter
  PoD                      = 1                # pitch to diameter ratio for parallel bundle
  SC_HTC                   = 1                # Sensitivity coefficient for HTC,
                                              # multiplicative
  SC_WF                    = 1                # Sensitivity coefficient for wall friction,
                                              # multiplicative
  WF_geometry_type         = Pipe             # wall friction geometry type
  WF_user_option           = Default          # user-option for wall friction model
  axial_mixing             = 0                # If the 1-D axial mixing model is activated
  component_type           = PBOneDFluidComponent # The type of the component
  end_elems_refinement     = 1                # number of element for the end element
                                              # in this OneDComp
  eos                      = (required)       # The name of EOS to use
  f                        = (no_default)     # friction
  fluid_conduction         = (no_default)     # if modeling the fluid axial conduction
  heat_source              = 0.               # Volumetric heat source
  initial_P                = (no_default)     # Initial pressure in the OneDComp
  initial_PS               = (no_default)     # Initial value of passive scalar
                                              # in the OneDComp
  initial_T                = (no_default)     # Initial temperature in the OneDComp
  initial_V                = (no_default)     # Initial velocity in the OneDComp
  inlet_area_ratio         = 1                # Volume area over inlet (jet) area
  input_parameters         = (no_default)     # Name of the ComponentInputParameters
                                              # user object
  lam_factor               = 1                # a user-input shape factor for laminar
                                              # friction factor for non-circular
                                              # flow channels
  length                   = (required)       # Length of the OneDComp
  n_elems                  = (required)       # number of element in this OneDComp
  n_layers_coolant         = (no_default)     # Number of layers in the coolant channel
  offset                    = '0 0 0'          # Offset of the origin for mesh generation
  orientation               = '0 0 1'          # Orientation vector of the component
  position                  = '0 0 0'          # Origin (start) of the component
  rotation                  = 0                # Rotation of the component (in degrees)
  roughness                 = 0                # roughness, [m]
  scalar_source             = (no_default)     # Volumetric scalar source
  scaling_velocity          = (no_default)     # a user-input global velocity for PSPG
                                               # scheme
  tao_pspg                  = (no_default)     # tao_pspg
  tao_supg                  = (no_default)     # tao_supg
  turb_factor               = 1                # a user-input shape factor for turbulent
                                               # friction factor for non-circular
                                               # flow channels
  type                      = PBOneDFluidComponent

  User_defined_HTC_parameters = '0 0 0 0 0 0 0'  # User-defined HTC model parameters
  User_defined_WF_parameters  = '0 0 0'          # User-defined WF model parameters
  coolant_density_reactivity_feedback = 0        # Enable coolant density reactivity
                                                 # feedback.

  coolant_reactivity_coefficients      = (no_default)  # Coolant reactivity coefficients
                                                       # (delta_k / k per kg)
  coolant_reactivity_coefficients_fn   = (no_default)  # Coolant reactivity
                                                       # coefficients function.
[../]
```



(a) A round pipe



(b) A pipe with irregular cross section

Figure 4.1: SAM PBOneDFluidComponent examples.

Each of the input parameters are discussed as follows. Geometry-related input parameters are discussed first,

- `A` (required)

  Cross-sectional (flow) area of the flow channel. For example, for round pipes, it is simply $\pi D^2/4$, see figure 4.1.

- `length` (required)

  Length of the flow channel, see figure 4.1.

- `position`

The origin of the one-dimensional pipe, in (x0, y0, z0), see figure 4.1. The default value is (0, 0, 0), i.e., `position = '0 0 0'`.

- `orientation`

  The orientation vector of the one-dimensional pipe, in (dx, dy, dz), see figure 4.1. Note that it does not have to be a unit vector. The default value is (0, 0, 1), i.e., `orientation = '0 0 1'`.

- `n_elemes (required)`

  Number of elements used for the component in the axial direction.

- `Dh (required)`

  Hydraulic diameter of the flow channel. For round pipes, it is simply the pipe diameter; while for flow channels with irregular shape of cross section, it is calculated as:

  $$D_h = \frac{4A}{P}$$

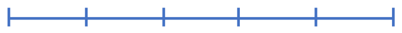
  where A is the cross-sectional area, and P is the wetted perimeter, see figure 4.1.

- `rotation`

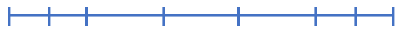
  Rotation of the component (in degrees), which will be used to construct displaced mesh within the code. This is related how SAM internally builds and handles meshes. The default value of this input parameter is 0, and in most cases, it is safe to leave it unspecified.

- `end_elems_refinement`

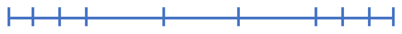
  Number of refined elements for the end elements at the begin and end of this component. The default value is 1, and therefore no refinement. Several examples are shown in figure 4.2 to illustrate how `end_elems_refinement` works.



(a) `n_elemes`=5, `end_elems_refinement` = 1 (default)



(b) `n_elemes`=5, `end_elems_refinement` = 2



(c) `n_elemes`=5, `end_elems_refinement` = 3

Figure 4.2: PBOneDFluidComponent with end element refinements.

- `offset`

This parameter accepts an offset, in (dx, dy, dz), from its origin point, i.e., `position`, such that the true origin point of the flow channel becomes (origin + offset). Its default value is (0, 0, 0), meaning no offset at all. This parameter will be depreciated as SAM moves into the real space, instead of the displayed mesh system it is currently using.

Input parameters related to equation of state, and local initial conditions are given as follows:

- `eos (required)`

  The name of equation of state to be used in this component.

- `initial_P`, `initial_V`, and `initial_T`

  Local initial condition for pressure, fluid velocity, and temperature, respectively. If specified, these values will override those specified in the global parameter list, and will be used to initialize pressure, fluid velocity, and temperature of this component. If not specified, those global initial values will be used.

- `initial_PS`

  Local initial conditions for passive scalars. If specified, they override values specified in the global parameter list. If not specified, the global initial values will be used.

The following input parameters are related to how wall frictional coefficients will be calculated in the fluid component,

- `f`

  A user-specified constant wall frictional coefficient. If not provided, the wall frictional coefficient will be automatically calculated within the code, see section 4.3 of SAM Theory Manual [1]. Whenever provided, this input parameter will shadow all other wall-friction-related input parameters, such as, `WF_user_option`, i.e., they will all simply be ignored.

- `roughness`

  Wall roughness. Some wall friction correlations, e.g., the Churchill correlation, require the wall roughness to compute the frictional coefficient. The default value is 0 m.

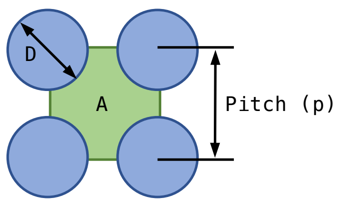
- `WF_geometry_type`

  Geometry type for SAM to select appropriate wall friction correlations. Currently, there are four types of geometries for selection: ‘Pipe’ (default), ‘WireWrap’, ‘SquareLattice’, and ‘Plate’, among which, ‘WireWrap’ is typical for sodium fast reactor designs, and ‘SquareLattice’ is typical for light water reactor designs.

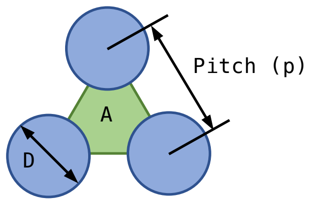
- `WF_user_option`

  Users can also directly specify wall friction correlations to be used to compute the frictional coefficient, however, it should be noted that some correlations only work with certain geometry type, `WF_geometry_type`.

  The available options for this parameters are: ‘Default’, ‘BlasiusMcAdams’, ‘ZigrangSylvester’, ‘Churchill’, ‘ChengTodreas’, and ‘User’.



(a) Square-lattice fuel bundle



(b) Hexagonal-lattice fuel bundle

Figure 4.3: Fuel bundles in (a) square-lattice, typically seen in light water reactor designs; and (b) hexagonal-lattice, typically seen in sodium fast reactor designs.

First, if the 'User' option is selected, SAM will compute the wall frictional coefficient from the following Reynolds number-dependent correlation:

$$f = A + B \times \text{Re}^C$$

and SAM is also expecting an additional input parameter, `User_defined_WF_parameters`, in which the user-specified constants are given as 'A B C'. This user-specified correlation is to be used in both the laminar and turbulence flow regimes.

For options other than 'User', 'Default' and 'BlasiusMcAdams' are effectively identical: for laminar flow, the Darcy's model will be used, and for turbulent flow, the Blasius correlation is used for Reynolds number smaller than 3×10^4, and the McAdams correlation for Reynolds number larger than 3×10^4.

The 'Churchill' option will use the Churchill model for wall friction coefficient in both the laminar and turbulent flow regimes.

When 'ZigrangSylvester' option is selected, the Zigrang-Sylvester correlation will be used for the turbulent flow regime, while for the laminar flow, the Darcy's model will be used.

When 'ChengTodreas' option is selected, the Cheng-Todreas correlation will be used for both the laminar and turbulent flow regimes. It is also the default option when 'WireWrap' type of geometry is specified, i.e., `WF_geometry_type = WireWrap`.

Users are referred to section 4.3 of the SAM Theory Manual [1] for more details of the wall friction correlations.

- `User_defined_WF_parameters`

  As discussed in `WF_user_option`, when `WF_user_option = User`, this input parameter accepts a set of three values for 'A B C' to compute use-provided wall friction factor. If `WF_user_option = User`, this input parameter is expected from user input. The default values are '0 0 0'.

- `PoD`

  This parameter defines the pitch (p) to diameter (D) ratio in rod bundles, see figure 4.3. This ratio is to be used to compute wall friction factor in, for example, the Cheng-Todreas correlation, and convective heat transfer coefficient in, for example, the Kazimi-Carelli correlation.

- `HoD`

This parameter defines ratio of “wire lead length” (H) to rod diameter (D), see figure 4.4. Currently, this parameter is only used in the Cheng-Todreas correlation to compute wall friction factor in the wire-wrapped fuel bundle geometry.

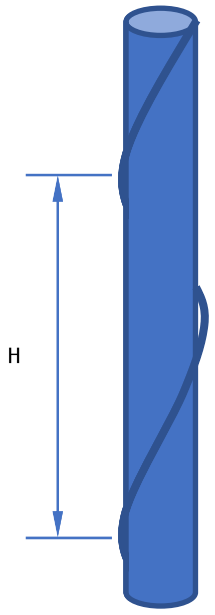


Figure 4.4: Typical SFR wire-wrapped rod configuration.

- `lam_factor` and `turb_factor`

  A user-input shape factor for laminar/turbulent flow friction factor for non-circular flow channels. Their default values are both 1.0. Basically, they work as multipliers that are multiplied to the values computed from wall friction correlations other than user-specified constant wall frictional coefficient `f` and user-specified Reynolds number-dependent correlation `WF_user_option`.

- `SC_WF`

  This is the same wall friction coefficient multiplier parameter as defined in the global parameter list, section 4.1. If specified in this component, it will override the globally defined parameter locally, i.e., in this component.

The following input parameters are related to wall heat transfer,

- `Hw`

  A user-specified constant wall heat transfer coefficient. If not provided, the wall heat transfer coefficient will be automatically calculated within the code, see section 4.2 of SAM Theory Manual [1]. Whenever provided, this input parameter will shadow all other wall-heat-transfer-related input parameters, such as, `HTC_user_option`, i.e., they will all simply be ignored.

- `Ph`

  This parameter is the heated perimeter. If heat transfer takes place on the entire wetted perimeter, the heated perimeter is the same as the wetted perimeter, see figure 4.1. For fuel bundles shown in figure 4.3, assuming all fuel rods are heated, the heated perimeters are πD and $\pi D/2$ for (a) square-lattice and (b) hexagonal-lattice fuel bundles, respectively. However, it is not always true that the heated perimeter is the same as the wetted perimeter. If, for example, one of the rod in 4.3 (a) is unheated, the heated perimeter is $3\pi D/4$, instead of πD, which is the value of wetted perimeter.

This is an optional input parameter without a default value given. If specified, it will be used to compute the heat transfer area density, see `HT_surface_area_density`.

- `HT_surface_area_density`

  This parameter accepts user-specified value for heat transfer surface area density, a_w, which is heat transfer surface area per fluid volume [m^2/m^3]. In most cases, it is computed as the ratio of heated perimeter to cross-sectional flow area,

  $$a_w = \frac{P_h}{A}$$

  For a round pipe, as shown in figure 4.1 (a), it is:

  $$a_w = \frac{P_{heated}}{A} = \frac{\pi D}{\pi D^2/4} = \frac{4}{D}$$

  For fuel bundles, as for example shown in figure 4.3 (b), it is:

  $$a_w = \frac{P_{heated}}{A} = \frac{3 \times \frac{\pi D}{6}}{A}$$

  Care should be taken when providing this parameter, as it often depends on how input model is set up. If not specified correctly, often energy imbalance between fluid components and heat structures would be introduced.

  As for user input, this is an optional input parameter without a default value given. If the heated perimeter, `Ph`, is specified, heat transfer area density is computed from its definition, $a_w = P_h/A$. Users can also specify a constant value for a_w. If neither heated perimeter nor this parameter is given, it is automatically assumed that the heated perimeter is the same as the wetted perimeter, and thus:

  $$a_w = \frac{P_h}{A} = \frac{P}{A} = \frac{4}{D_h}$$

  in which P is the wetted perimeter.

- `HTC_geometry_type`

  Geometry type for SAM to select appropriate heat transfer coefficient correlations. There are four types channel geometries available in SAM, “Pipe (default)”, “Bundle”, “Vertical-Plate”, and “Horizontal-Plate”[1].

- `HTC_user_option`

  Similar to wall friction correlation, users can also directly specify correlations to compute heat transfer coefficient. The available options for this parameters are: ‘Default’, ‘NotterSleicher’,

[1] Vertical-Plate and Horizontal-Plate have not been treated yet.

'Aoki', 'ChengTak', 'Mikityuk', 'ModifiedSchad','GraberRieger', 'McAdams', 'ChurchillChu', 'GaddisGnielinski', 'UserForced' and 'UserNatural'.

If this input parameter is not specified, SAM goes to 'default' options to select appropriate heat transfer coefficient correlations depending on combination of heat transfer geometry, fluid type (liquid metal or not), and flow condition (laminar or turbulent). For pipe geometry, the default correlation for liquid metal ($\mathrm{Pr} < 0.1$) is the Seban-Shimazaki correlation (see section 4.2 of SAM Theory Manual [1]); for fluids other than liquid metal, SAM picks the largest value among those computed from the Dittus-Boelter correlation, the Churchill-Chu correlation, and the correlation for forced laminar flow (Nu = 4.36). For fuel bundle geometry, the default correlation for liquid metal is the same as used for pipe geometry. For non-liquid metal fluids, SAM picks the largest value among those computed from the Inayatov model (modified Dittus-Boelter correlation for fuel bundle geometry), the Churchill-Chu correlation, and the correlation for forced laminar flow (Nu = 4.36).

For pipe geometry, users can also select one of the following correlations, 'NotterSleicher', 'Aoki' or 'ChengTak' for liquid metal, or one from 'McAdams' and 'ChurchillChu' for non-liquid metal fluids. For fuel bundle geometry, available options are 'Mikityuk', 'ModifiedSchad', and 'Graber-Rieger' for liquid metal; and 'McAdams', 'ChurchillChu', and 'GaddisGnielinski' for non-liquid metal fluids.

SAM also allows users to specify user-defined correlations. Users could select the 'UserForced' option (for forced convection), and then specify a set of 7 numbers, i.e.,

$$[\mathrm{Nu}_0, a, b, c, d, e, f]$$

in the `User_defined_HTC_parameters` input parameter, which will be used to compute the Nusselt number in the form of:

$$\mathrm{Nu} = \mathrm{Nu}_0 + a\left(\mathrm{Re}^b + c\right)\mathrm{Pr}^d\left(1 + e\mathrm{Re}^f\right)^{0.1}$$

Users could also select the 'UserNatural' option (for natural convection), and then specify a set of 3 numbers in the `User_defined_HTC_parameters` input parameter,

$$[\mathrm{Nu}_0, a, b]$$

which will be used to compute the Nusselt number in the form of:

$$\mathrm{Nu} = \mathrm{Nu}_0 + a\mathrm{Ra}^b$$

For heat transfer coefficients, users are referred to SAM theory manual [1] for more details.

- `User_defined_HTC_parameters`

  This input parameter expects either a set of 7 numbers when `HTC_user_option = UserForced`, or a set of 3 numbers when `HTC_user_option = UserNatural` (see the previous item). The default values are '0 0 0 0 0 0 0'.

- `SC_HTC`

  This is the same heat transfer coefficient multiplier parameter as defined in the global parameter list, section 4.1. If specified in this component, it will override the globally defined parameter locally, i.e., in this component.

Input parameters related to reactivity feedback model are given as follows:

- `coolant_density_reactivity_feedback`

  If specified true, this input parameter enables coolant density reactivity feedback. By default, it is false.

- `n_layers_coolant`

  This parameter specifies the number of layers of coolant in the flow channel. In combination of `coolant_reactivity_coefficients` or `coolant_reactivity_coefficients_fn`, the average coolant density in each of these layers will be used to compute the total reactivity feedback in this flow channel. If not specified, it takes the value of number of elements, i.e., `n_elems`.

- `coolant_reactivity_coefficients`

  This parameter specifies a list of coolant reactivity coefficients. If there is only one value in this list, this value will be used in all layers of coolant to compute total reactivity feedback. Otherwise, the total number of values in this list should be equal to number of layers, i.e., `n_layers_coolant` (if specified) or `n_elems`.

- `coolant_reactivity_coefficients_fn`

  The parameter specifies a function (name) to be used to compute coolant reactivity coefficients. The function should be spatially distributed along the channel's axial direction. The reactivity coefficient will be sampled in the middle point of each layer of coolant.

All other input parameters are discussed as follows:

- `tao_supg`

  An optional input parameter to accept user-specified SUPG stabilization parameter, τ_{SUPG}. If not specified, τ_{SUPG} is automatically computed within the code. It is not recommended to specify this parameter.

- `tao_pspg`

  An optional input parameter to accept user-specified PSPG stabilization parameter, τ_{PSPG}. If not specified, τ_{PSPG} is automatically computed within the code. It is not recommended to specify this parameter.

- `scaling_velocity`

  An optional input parameter to accept user-specified reference velocity (magnitude) to compute PSPG stabilization parameter, τ_{PSPG}. If not specified, SAM automatically picks appropriate velocity magnitude to compute τ_{PSPG}. It is not recommended to specify this parameter.

- `fluid_conduction`

  This input parameter overrides the one specified in the global parameter list, which specifies if axial heat conduction effect of the fluid should be included in the fluid energy equation.

- `heat_source`

  This input parameter specifies a direct volumetric heating source to the fluid. A number can be simply specified to assign a constant value as the volumetric heating source. A function name, which must have been given in the `[Function]` input block, can also be given to this input parameter, so the volumetric heating source will be calculated from this given function.

- `scalar_source`

  This input parameter specifies a list of volumetric sources to the passive scalar variables. Similar to `heat_source`, both numbers and function names are acceptable options. In addition, numbers and function names could be mixed in the same list.

- `axial_mixing`

  This input parameter specifies if the one-dimensional axial mixing model should be activated. The default value is false, i.e., axial mixing model is not activated.

- `inlet_area_ratio`

  This input parameter specifies the ratio of volume area to inlet (jet) area for the one-dimensional axial mixing model. The default value is 1.0.

- `Cgv`

  This input parameter specifies the mixing coefficient due to velocity and geometry effects for the one-dimensional axial mixing model. The default value is half of the `inlet_area_ratio` value.

- `Cgb`

  This input parameter specifies the mixing coefficient due to buoyancy and geometry effects for the one-dimensional axial mixing model. The default value is 1.0.

- `input_parameters`

  This input parameter is designed to allow SAM input components share common features. For example, in a flow loop consisting of many pipes of the same type, this input parameter allows that these common features (e.g., flow area, hydraulic diameter, etc.) are to be inputted for only once. The details are provided in section 4.5.

An example input block is given as follows:

```
[Components]
  ......
  [./pipe1]
    type = PBOneDFluidComponent
    eos = eos                     # The equation-of-state
    position = '0 0 0'            # The origin of this component
    orientation = '0 0 1'         # The orientation of the component
    A = 0.01                      # Area of the One-D fluid component
    heat_source = 0               # Volumetric heat source
    f = 0.01                      # Specified friction coefficient
    Dh = 0.01                     # Equivalent hydraulic diameter
    length = 1                    # Length of the component
    n_elems = 100                 # Number of elements used in discretization
  [../]
[]
```

4.3.2 HeatStructure

HeatStructure is the most basic solid structure component in SAM. It represents a unit one-D or two-D component in Cartesian or cylindrical coordinates to simulate the heat conduction in solid structures. The geometry parameters such as the thickness and length are provided in the input file. Temperature-dependent solid material properties can be provided in tabular or functional form user-supplied data. Internal volumetric heating can be specified by the user input. Input parameters of HeatStructure is given as follows:

```
[./HeatStructure]
  Ts_init              = (no_default)  # Initial temperature
  axial_offset         = 0             # Axial offset for cylindrical heat structures
  depth_plate          = (no_default)  # depth of plate in case of plate geometry.
                                       # will be used to calculate the volume.
  dim_hs               = 2             # Dimension of the geometry (1 = 1D, 2 = 2D)
  elem_number_axial    = 1             # Number of axial elements of heat structure
  elem_number_radial   = (required)    # Number of radial elements of heat structure
  end_elems_refinement = 1             # number of element for the end element
                                       # in this Component
  heat_source_solid    = 0             # heat source in solid
  hs_names             = (no_default)  # User given heat structure names
  hs_power             = (no_default)  # total power in the heat structure.
  hs_power_shape_fn    = (no_default)  # axial power shape of the heat structure.
  hs_type              = plate         # Geometry type of the heat structure
  input_parameters     = (no_default)  # Name of the ComponentInputParameters user object
  length               = (required)    # Length of the heat structure
  material_hs          = (required)    # Name of the material used in the heat structure
  offset               = '0 0 0'       # Offset of the origin for mesh generation
  orientation          = '0 0 1'       # Orientation vector of the component
  position             = '0 0 0'       # Origin (start) of the component
  power_fraction       = (no_default)  # fraction of total power goes into different blocks.
  radius_i             = (no_default)  # the radius of the inner wall of the heat
                                       # structure, needed when the hs is a cylinder
  rotation             = 0             # Rotation of the component (in degrees)
  width_of_hs          = (required)    # Width of heat structure
[../]
```

Each of these input parameters are discussed as follows:

- position, orientation, rotation, offset

  These input parameters are defined the same way as discussed in section 4.3.1. Also, see figure 4.5 for reference.

- dim_hs

  It specifies how the heat structure is modeled, either in one-dimensional (dim_hs = 1) or two-dimensional (dim_hs = 2). The default and recommended value is 2, i.e., two-dimensional.

- Ts_init

  The initial temperature for the heat structure. If not specified, it seeks the global initial temperature (see global_init_T in section 4.1) as the initial temperature.

- hs_type

  Geometry type of the heat structure, which can be either of 'plate' (default) or 'cylinder' type. Note that this input parameter is case sensitive, e.g., 'Plate' is not equivalent to 'plate'. An example 2D plate type of heat structure is shown in figure 4.5.

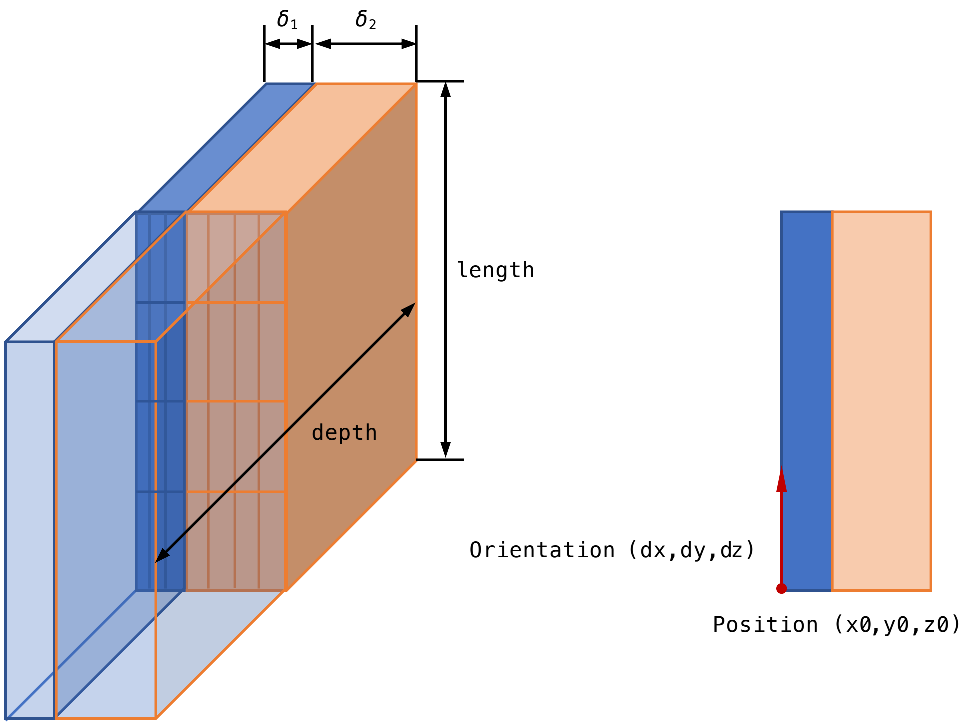


Figure 4.5: An example of two-dimensional plate type of heat structure.

- `hs_names`

  This input parameter specifies a vector of names for each layer of heat structure. If not specified, SAM automatically creates names for each layer of heat structure. For example, in figure 4.5, the automatically generated names for the two layers would be: `<HS name>:hs0` and `<HS name>:hs1`.

- `elem_number_axial`

  Number of elements in the axial direction (along the length direction, see 4.5) of 2-D heat structure, or number of intervals between 1-D heat structures if it is 1-D heat structure. The default value is 1. In figure 4.5, `elem_number_axial = 4`.

- `elem_number_radial` (required)

  This input parameter accepts a vector of numbers that specify the number(s) of elements to be used for each layer of heat structure in the wall-thickness direction. As for example, in figure 4.5, `elem_number_radial = '3 4'`.

- `width_of_hs` (required)

  This input parameter specifies a vector of thickness for the layer(s) of heat structure. The size of this input vector should be the same as `n_wall_elems`. As for example, in figure 4.5, `width_of_hs = ''$\delta _1$ $\delta _2$'`.

- `material_hs` (required)

This input parameter specifies a vector of heat structure material name(s) for the layer(s) of heat structure, for example, `material_wall = 'SS-304 Wall-Material-2'`.

- `heat_source_solid`

  As one of user-specified heat source input options, this input parameter accepts a constant number that is used to specify a uniformly distributed constant volumetric heat source (W/m^3) in the entire heat structure. The default value is 0. If more complex heat source input than this simple constant value is desired, SAM provides other input options, see `hs_power`.

- `hs_power` and `power_fraction`

  `hs_power` specifies the total power, in [W], of the heat structure. If `power_fraction` is not further specified, it is assumed that the total power is uniformly distributed on the entire heat structure, and therefore, the volumetric heat source is calculated as total power divided by total volume of the heat structure.

  In case that power is not uniformly distributed, `power_fraction` accepts a vector of values that specifies the fraction of the total power for each layer of heat structure. The size of this vector has to be the same as the number of layers in the heat structure. For example, in figure 4.5, one could specify `hs_power = 1000` and `power_fraction = '0.9 0.1'`, and thus 90% of the total power goes to the first layer (to the left), and 10% goes to the second layer (to the right). Volumetric heat source in each layer of heat structure is then calculated as power in this heat structure layer divided by the solid volume of this heat structure layer.

- `hs_power_shape_fn`

  This input parameter accepts a function name, which can be a function of time and/or space. It is important to note that the function value, which could be both temporal and spatial dependent, is multiplied to local volumetric heat source (see previous two items), and there is no re-normalization of total power.

- `depth_plate`

  This input parameter is only required when `hs_power` is specified and the heat structure type is "plate". It is required to compute the volume of each heat structure layer. For example, in figure 4.5, the volume of the first heat structure layer is calculated as: Length $\times \delta_1 \times$ depth.

- `radius_i`

  This input parameter specifies the inner radius of the left-most wall if the heat structure type is cylinder, see figure 4.7 as an example. The default value is 0, if not specified.

- `end_elems_refinement`

  Number of refinement for the end elements in the beginning and ending of the component (in the axial direction). The default value is 1. It is only available when the heat structure dimension is two. It usage is similar to the same input parameter for `PBOneDFluidComponent`, see section 4.3.1, and it is normally used in pair with `PBOneDFluidComponent` in `PBHeatExchanger` (see section 4.3.15).

- `axial_offset`

  Axial offset for cylindrical heat structures

- input_parameters (advanced)

  This parameter is similar to that of PBOneDFluidComponent, also see section 4.5.

4.3.3 PBPipe

In SAM, PBPipe is directly inherited from PBOneDFluidComponent, with the concept to model a one-dimensional pipe flow and its pipe wall with one layer (or several layers) of HeatStructure, as illustrated in figure 4.6. Its input parameters are therefore a superset of input parameters of PBOneDFluidComponent and those to define wall heat structures.

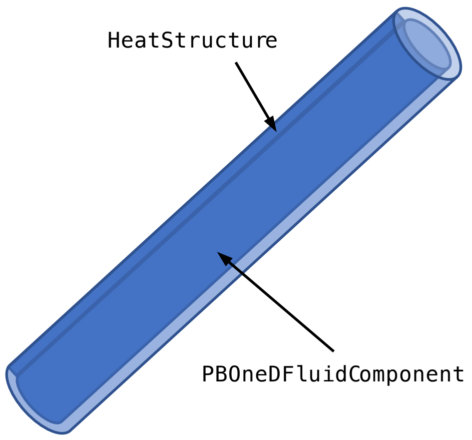


Figure 4.6: SAM's PBPipe component, which consists of a PBOneDFluidComponent to model the one-dimensional fluid flow and one layer (or several layers) of HeatStructure to model its wall.

The input parameter subset for PBOneDFluidComponent has been discussed in section 4.3.1. In this section, only the input parameters to define wall heat structures are discussed:

```
[./PBPipe]
  # Input parameters inherited from PBOneDFluidComponent are not listed.

  HS_BC_type          = Adiabatic      # Heat structure boundary condition type
  T_amb               = 300            # ambient temperature
  T_wall              = 600            # Fixed Temperature BC at the outer pipe wall surface
  Twall_init          = (no_default)   # Initial wall temperature
  dim_wall            = 2              # The dimension of the mesh used for the wall:
                                       # 1 = 1D, 2 = 2D (default).
  disp_mode           = 1              # 1.0 for +y display, -1.0 for -y display.
                                       # More complicated display modes are necessary
  h_amb               = (no_default)   # convective heat transfer coefficient with ambient
  heat_source_solid   = (no_default)   # heat source in solid
  hs_type             = plate          # Geometry type of the heat structure
  material_wall       = (required)     # Name of the material used in the wall
  n_wall_elems        = (required)     # number of elements in the wall
  name_comp_right     = (no_default)   # The name of the right liquid volume
                                       # connected to the heat structure
  qs_wall             = (no_default)   # Heat flux at the outer pipe wall surface
  radius_i            = (no_default)   # the radius of the inner pipe wall
  wall_thickness      = (required)     # Thickness of the wall
  input_parameters    = (no_default)   # Name of the ComponentInputParameters
                                       # user object
[../]
```

The detailed descriptions of these heat structure-related input parameters are given as follows:

- `Twall_init`

  The initial wall temperature for heat structures. If not specified, it first seeks local initial fluid temperature (see `initial_T` in section 4.3.1) as initial wall temperature; if local initial fluid temperature is neither given, it then seeks the global initial temperature (see `global_init_T` in section 4.1) as the initial wall temperature.

- `dim_wall`

  The same as `dim_hs` in section 4.3.2, it specifies how the wall heat structures are modeled, either in one-dimensional (`dim_wall = 1`) or two-dimensional (`dim_wall = 2`). The default and recommended value is 2, i.e., two-dimensional.

- `hs_type`

  Geometry type of the heat structure, which can be either of 'plate' (default) or 'cylinder' type. Note that this input parameter is case insensitive, e.g., 'Plate' is equivalent to 'plate'. This is the same as `hs_type` in section 4.3.2.

- `radius_i`

  This input parameter specifies the inner radius of the pipe wall, if a cylinder type of heat structure(s) is used to model pipe wall, see figure 4.7. If not specified, it takes half of the hydraulic diameter value, i.e., $D_h/2$.

- `n_wall_elems (required)`

  This input parameter accepts a vector of numbers that specify the number(s) of elements to be used for each layer of heat structure in the wall-thickness direction. As for example, in figure 4.7, `n_wall_elems = '2 3'`. This parameter is the same as `elem_number_radial` in section 4.3.2.

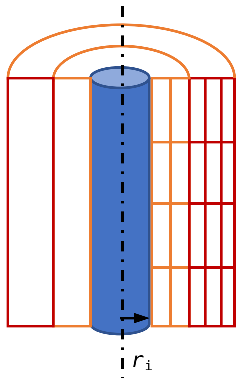


Figure 4.7: An input example of PBPipe with two layers of heat structures to model its wall. For example, it could represent a layer of metal wall and an extra layer of thermal insulation material.

- `wall_thickness (required)`

  This input parameter specifies a vector of wall thickness for the layer(s) of wall heat structure. The size of this input vector should be the same as `n_wall_elems`. This parameter is the same as `width_of_hs` in section 4.3.2.

- `material_wall` `(required)`

  This input parameter specifies a vector of heat structure material name(s) for the layer(s) of wall heat structure, for example, `material_wall = 'SS-304 Wall-Material-2'`. For obvious reason, the size of this input vector should be the same as `n_wall_elems`. This parameter is the same as `material_hs` in section 4.3.2.

- `heat_source_solid`

  This input parameter specifies a vector of volumetric heat source (in numbers) of wall heat structures. The vector size has to be the same as `n_wall_elems`.

- `HS_BC_type`

  This input parameter specifies the boundary condition type for the pipe outer wall surface. Available options for this parameter are: "Adiabatic (default)", "Temperature", "Convective", and "Coupled".

  "Adiabatic", as its name suggests, sets an adiabatic boundary condition for the pipe outer wall surface.

  "Temperature" sets a Dirichlet temperature boundary condition for the pipe outer wall surface. When this boundary condition type is specified, the Dirichlet temperature boundary condition value is also expected from the input file (see `T_wall`).

  "Convective" sets a convective boundary condition to model heat transfer between the pipe outer wall surface and the ambient. Additional input parameters are to be supplied for "Convective" type of boundary condition. This boundary condition type could be supplemented by another two input parameters: user-specified ambient temperature (see `T_amb`) and user-specified heat transfer coefficient (see `h_amb`). In addition, a user-specified wall heat flux could be directly given on the pipe outer wall surface (see `qs_wall`).

  "Coupled" sets a conjugate heat transfer boundary condition for the pipe outer wall surface. In this case, the volume component, with which the outer surface transfers heat, has to be specified in the `name_comp_right`.

- `T_wall`

  Pipe outer wall surface temperature in case that "Temperature" is specified for `HS_BC_type`. It has to be a number, and the default value for this input parameter is 600 K.

- `T_amb` and `h_amb`

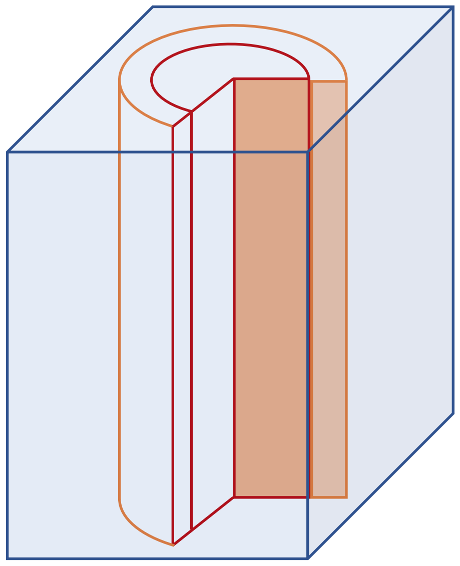
  When "Convective" is specified for `HS_BC_type`, `T_amb` accepts user-specified ambient temperature, and `h_amb` accepts user-specified heat transfer coefficient. Both input parameters accept either a number or a function name. The default value for `T_amb` is 300 K.

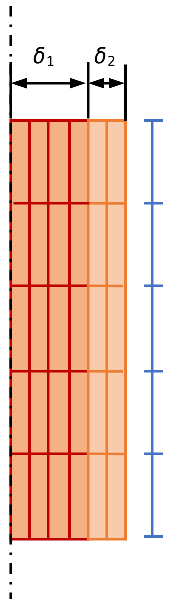
- `qs_wall`

  When "Convective" is specified for `HS_BC_type`, besides `T_amb` and `h_amb`, a user-specified wall heat flux could be directly given to the pipe outer wall surface. It can be either a number of a function name.

- `input_parameters`

  See section 4.5.


(a)



(b)

Figure 4.8: PBCoreChannel component (a) SAM's PBCoreChannel component simulates the average coolant flow in rod bundles and heat conduction inside a fuel rod; and (b) An example mesh used in the PBCoreChannel component, 2-D mesh for heat structure and 1-D mesh for fluid flow.

4.3.4 PBCoreChannel

PBCoreChannel simulates the average coolant flow in rod bundles and heat conduction inside a fuel rod, as well as the convective heat transfer between the coolant and the fuel rod. It is composed of a PBOneDFluidComponent and a HeatStructure. This is also the so-called "Single-Channel" approach to model the fuel assembly. Axial power profiles and the power fractions of total reactor power can be specified for the component. If an outer structure (duct wall) is added to PBCoreChannel, it becomes PBDuctedCoreChannel, which simulates the ducted fuel assemblies as those in SFRs.

When more complex PBCoreChannel is needed to model the reactor fuel assemblies having different axial regions, FuelAssembly or DuctedFuelAssembly are provided in SAM.

From thermal-hydraulics point of view, PBCoreChannel is quite similar to PBPipe, both of which consist of a 1-D fluid flow model and a heat structure, although PBPipe assumes heat structures to be pipe walls, while PBCoreChannel assumes heat structures to be fuel rods, figure 4.8. The major difference between these two components is that PBCoreChannel has a built-in interface to interact with ReactorPower component. It receives power as heat source from the ReactorPower component, and also provides the capability to model reactivity feedback. The three types of reactivity feedback mechanisms include fuel's Doppler effect, fuel expansion effect, and coolant density effect (via PBOneDFluidComponent).

The full list of input parameters are given in the following table. Most of them are the same as those for PBOneDFluidComponent (section 4.3.1) or HeatStructure (section 4.3.2).

```
[./PBCoreChannel]
  A                      = (required)    # See PBOneDFluidComponent
  Dh                     = (required)    # See PBOneDFluidComponent
```

```
HTC_geometry_type     = Pipe           # See PBOneDFluidComponent
HTC_user_option       = Default        # See PBOneDFluidComponent
HoD                   = 1              # See PBOneDFluidComponent
Hw                    = (no_default)   # See PBOneDFluidComponent
Ph                    = (no_default)   # See PBOneDFluidComponent
PoD                   = 1              # See PBOneDFluidComponent
SC_HTC                = 1              # See PBOneDFluidComponent
SC_WF                 = 1              # See PBOneDFluidComponent
end_elems_refinement  = 1              # See PBOneDFluidComponent
eos                   = (required)     # See PBOneDFluidComponent
f                     = (no_default)   # See PBOneDFluidComponent
fluid_conduction      = (no_default)   # See PBOneDFluidComponent
initial_P             = (no_default)   # See PBOneDFluidComponent
initial_PS            = (no_default)   # See PBOneDFluidComponent
initial_T             = (no_default)   # See PBOneDFluidComponent
initial_V             = (no_default)   # See PBOneDFluidComponent
n_elems               = (required)     # See PBOneDFluidComponent
WF_geometry_type      = Pipe           # See PBOneDFluidComponent
WF_user_option        = Default        # See PBOneDFluidComponent
lam_factor            = 1              # See PBOneDFluidComponent
length                = (required)     # See PBOneDFluidComponent
orientation           = '0 0 1'        # See PBOneDFluidComponent
position              = '0 0 0'        # See PBOneDFluidComponent
rotation              = 0              # See PBOneDFluidComponent
roughness             = 0              # See PBOneDFluidComponent
scalar_source         = (no_default)   # See PBOneDFluidComponent
scaling_velocity      = (no_default)   # See PBOneDFluidComponent
tao_pspg              = (no_default)   # See PBOneDFluidComponent
tao_supg              = (no_default)   # See PBOneDFluidComponent
turb_factor           = 1              # See PBOneDFluidComponent
HT_surface_area_density      = (no_default)    # See PBOneDFluidComponent
User_defined_HTC_parameters = '0 0 0 0 0 0 0' # See PBOneDFluidComponent
User_defined_WF_parameters  = '0 0 0'         # See PBOneDFluidComponent
n_layers_coolant                     = (no_default) # See PBOneDFluidComponent
coolant_density_reactivity_feedback = 0            # See PBOneDFluidComponent
coolant_reactivity_coefficients     = (no_default) # See PBOneDFluidComponent
coolant_reactivity_coefficients_fn  = (no_default) # See PBOneDFluidComponent

Ts_init               = (required)     # See HeatStructure
dim_hs                = 1              # See HeatStructure
heat_source           = 0.             # See HeatStructure
material_hs           = (required)     # See HeatStructure
power_fraction        = (no_default)   # See HeatStructure
power_shape_function  = (no_default)   # See HeatStructure
width_of_hs           = (required)     # See HeatStructure

assembly_type              = RodBundle    # Fuel assembly geometry type
coupled_axial_expansion    = 0            # If using the displacement from
                                          # external thermo-mechanical module.
depth                      = 1            # The dimension of plate fuel in the
                                          # third direction, m
elem_number_of_hs          = (required)   # Number of elements of each heat structure
fuel_type                  = plate        # Geometry type of the fuel
mesh_disp_gap              = 0.005        # Mesh offset when creating heat structure
                                          # meshes
n_assemblies               = 1            # number of represented assemblies
n_heatstruct               = (required)   # Number of heat structures
name_of_hs                 = (required)   # User given heat structure names
n_rods                     = (no_default) # number of fuel rods per fuel assembly

eutectic_condition_expansion             = 1             # If using the free expansion
                                                         # model.
fuel_axial_expansion_reactivity_feedback = 0             # Enable fuel axial reactivity
                                                         # feedback.
```

```
  liquid_fuel                             = 0            # Enable fuel axial expansion
                                                         # reactivity feedback model
                                                         # for liquid fuel
  fuel_axial_expansion_reactivity_fn      = (no_default) # Axial reactivity function name.
  fuel_doppler_reactivity_coefficients    = (no_default) # Fuel Doppler reactivity
                                                         # coefficients
                                                         # (delta_k / k per kg)
  fuel_doppler_reactivity_coefficients_fn = (no_default) # Fuel Doppler reactivity
                                                         # coefficients
                                                         # (delta_k / k per kg)
  fuel_doppler_reactivity_feedback        = 0            # Enable fuel Doppler reactivity
                                                         # feedback.
  n_layers_axial_expansion                = (no_default) # Number of layers for fuel axial
                                                         # expansion reactivity feedback.
  n_layers_doppler                        = (no_default) # Number of layers in the fuel
                                                         # rod for fuel Doppler reactivity
                                                         # feedback.

  input_parameters    = (no_default)   # Name of the ComponentInputParameters
                                       # user object
[../]
```

- `n_assemblies`

  Number of assemblies grouped together that is represented by this `PBCoreChannel` component.

- `elem_number_of_hs` (required)

  Number of radial elements of heat structure. See `elem_number_radial` in section 4.3.2.

- `Ts_init` (required)

  The initial temperature for the heat structure, same as `Ts_init` in section 4.3.2, however this is a required input parameter for `PBCoreChannel` component.

- `n_heatstruct` (required)

  Number of heat structures. In some cases that no heat structures should be modeled in this component, specify zero to this parameter, and at the same time, specify "None" to `fuel_type`. When heat structure is modeled, a company component, `ReactorPower` is expected in the input file, which provides power to this component and accepts reactivity feedback from this component (optional).

- `name_of_hs` (required)

  This input parameter is similar to `hs_names` of `HeatStructure`, however, it is a required input parameter for this component. It accepts a vector of names, which specify the name of each heat structure of this component.

- `dim_hs`

  The dimension of the mesh used for the heat structure, same as `dim_hs` in section 4.3.2. However, the default value here is 1.

- `fuel_type`

  Same as `hs_type` in section 4.3.2. It can be "None" if no fuel rod is modeled.

- `depth`

  The depth of plate type of fuels, the same as `depth_plate` in section 4.3.2

- `end_elems_refinement`

  It is defined, but not used in this component.

- `mesh_disp_gap`

  This input parameter specifies mesh offset in the y-direction, with respect to the fluid component mesh, when creating heat structure meshes. The default value for this parameter is 0.005 [m].

- `n_rods`

  Number of fuel rods per fuel assembly. This parameter is only required when assembly type is “Block-Channel”.

- `fluid_conduction`

  It is defined, but not used in this component.

- `assembly_type`

  This parameter specifies the assembly type of this `PBCoreChannel` component. Available options include “RodBundle” (default), “Plates”, and “Block-Channel”. If “Block-Channel” is specified, an additional input parameter `n_rods` is expected.

- `fuel_axial_expansion_reactivity_feedback`

  This parameter specifies if reactivity feedback due to fuel axial expansion should be considered. By default, it is False (not considered). When it is specified True, additional input parameters are expected from user input (see discussion that follows).

- `liquid_fuel`

  This parameter enables the liquid fuel axial expansion reactivity feedback model. For the liquid fuel axial expansion reactivity, the fuel is assumed to expands freely in axial direction. Caution: this model works on the condition that user input fuel density is a function of fuel temperature. The density change in each axial layer contributes to the reactivity. Tips: for fuel beyond the fixed axial mesh, the user is expected to provide the fuel worth value through the `fuel_axial_expansion_reactivity_fn`.

- `n_layers_axial_expansion`

  When `fuel_axial_expansion_reactivity_feedback = True`, this parameter specifies the number of fuel layers in the axial direction (N_{layer}). The length of each of these layers is assumed to be the same, and thus it is equal to total fuel length divided by N_{layer}. If not specified, this parameter takes the number of fluid elements (the same as the number of heat structure elements in the axial direction) as its default value. Averaged axial displacement in fuel rod will be calculated in each of these layers for the component to compute the overall reactivity feedback due to fuel axial expansion.

- `fuel_axial_expansion_reactivity_fn`

  When `fuel_axial_expansion_reactivity_feedback = True`, this parameter specifies the name of a function that will be used to compute the overall reactivity feedback due to fuel axial expansion.

- `coupled_axial_expansion`

  When `fuel_axial_expansion_reactivity_feedback = True`, this parameter specifies that, if True, fuel axial expansion is computed from codes external to SAM; and if False (default), it will be calculated internally using SAM's built-in models.

- `eutectic_condition_expansion`

  When `fuel_axial_expansion_reactivity_feedback = True`, this parameter specifies that, if True (default), eutectic conditions are assumed for fuel and clad expansions; and if False, fuel and clad are assumed to expand freely.

- `fuel_doppler_reactivity_feedback`

  This parameter specifies if reactivity feedback due to fuel's Doppler effect should be considered. By default, it is False (not considered). When it is specified True, additional input parameters are expected from user input (see discussion that follows).

- `fuel_doppler_reactivity_coefficients_fn`

  When `fuel_doppler_reactivity_feedback = True`, this parameter accepts the name of a function that computes fuel's Doppler effect coefficients. The function can be spatial-dependent only. The value is evaluated in the middle of each fuel layer, see `n_layers_doppler`.

- `fuel_doppler_reactivity_coefficients`

  Instead of using a function, it is also possible to directly specify a vector Doppler effect reactivity coefficients for each fuel layer. This vector could contain only single value, such that it will be used for all fuel layers. Otherwise, the number of values in this vector shall be the same as `n_layers_doppler`.

- `n_layers_doppler`

  Similar to `n_layers_axial_expansion`, this input parameter specifies the number of fuel layers in the axial direction to compute fuel's Doppler effect, only needed when `fuel_doppler_reactivity_feedback = True`.

- `input_parameters`

  See section 4.5.

4.3.5 PBDuctedCoreChannel

`PBDuctedCoreChannel` is intended to model a fuel subassembly, which consists of a fuel bundle modeled as a PBCoreChannel and its duct wall modeled as an additional heat structure. Such a ducted fuel subassembly concept is typical in some sodium fast reactor designs. From user-input point of view, PBDuctedCoreChannel inherits all input parameters from the PBCoreChannel component (see 4.3.4), and requires additional input parameters to describe the duct wall, which are listed as follows:

```
[./PBDuctedCoreChannel]
  # Input parameters same as those in PBCoreChannel are not listed.

  name_of_duct          = duct          # User given duct wall heat structure names
  dim_duct              = 2             # Dimension of the geometry (1 = 1D, 2 = 2D)
  Tduct_init            = (no_default)  # Initial duct wall temperature
  duct_thickness        = (required)    # Thickness of the duct wall
  n_duct_elems          = (required)    # number of elements in the duct wall
  material_duct         = (required)    # Name of the material used in the duct wall
  disp_mode             = 1             # 1.0 for +y display, -1.0 for -y display.
                                        # More complicated display modes are necessary
  name_of_bpc           = (no_default)  # Adjacent BypassChannel names for the CoreChannel
  HT_surface_area_density_duct  = (required)  # duct side heating surface density
[../]
```

- `name_of_duct`

  This parameter specifies the name of the duct wall. If not specified, the default value is "duct".

- `dim_duct`

  Similar to `dim_hs` of `HeatStructure`, it specifies how duct wall heat structure is modeled, either in one-dimensional (1) or two-dimensional (2). The default and recommended value is 2, i.e., two-dimensional.

- `Tduct_init`

  This parameter specifies the initial temperature for the duct wall. If not specified, it takes the global initial temperature, `global_init_T` (see section 4.1), as the initial duct wall temperature.

- `duct_thickness (required)`

  The thickness of the duct wall.

- `n_duct_elems (required)`

  Number of elements to model the duct wall in its thickness direction.

- `material_duct (required)`

  This parameter specifies the duct wall material.

- `HT_surface_area_density_duct (required)`

  This parameter specifies the heat transfer surface area density of the duct wall with respect to the CoreChannel.

- `disp_mode`

  To be added.

- `name_of_bpc`

  This parameter specifies the name of BypassChannel adjacent for the CoreChannel.

4.3.6 PBBypassChannel

PBBypassChannel is just a PBOneDFluidComponent component with additional physics models. It is designed to model the bypass flow in the gaps between fuel assemblies. It includes the modeling of conjugate heat transfer with the neighboring fuel assembly duct walls. It can also model the direct coolant heating as a fraction of the total reactor power and using the same or different axial power shapes.

From user-input point of view, it inherits all input parameters from PBOneDFluidComponent, and requires additional input parameters to describe its neighboring PBDuctedCoreChannels and direct coolant heating.

```
[./PBBypassChannel]
  # Input parameters same as those in PBOneDFluidComponent are not listed.

  name_of_cc           = (no_default)  # Adjacent CoreChannel names
  power_fraction       = (no_default)  # fraction of reactor power goes into
                                       # this bypass channel
  power_shape_function = (no_default)  # axial power shape of the channel
  HT_surface_area_density_second = (no_default) # Heating surface density
[../]
```

- `name_of_cc`

  This input parameter specifies the names of the two adjacent PBDuctedCoreChannels.

- `power_fraction`

  This input parameter specifies the fraction of reactor power directly goes into this bypass channel. When specified, a `ReactorPower` component is also expected in the input file that provides the computation of reactor power.

- `power_shape_function`

  This input parameter accepts a function name, which can be a function of time and/or space. It is important to note that the function value, which could be both temporal and spatial dependent, is multiplied to local volumetric heat source, and there is no re-normalization of total power.

- `HT_surface_area_density_second`

  This input parameter is used in parallel wit `HT_surface_area_density` inherited from PBOneDFluidComponent. This parameter specifies a_w for the second PBDuctedCoreChannels in the list, while `HT_surface_area_density` specifies a_w for the first one.

4.3.7 PBMoltenSaltChannel

PBMoltenSaltChannel is a component intended to model the core behavior of molten-salt reactor designs. It is a PBOneDFluidComponent component with additional physics models that account for heating due to reactor power and decay curve.

From user-input point of view, it inherits all input parameters from PBOneDFluidComponent, and requires additional input parameters for the extra physics models.

```
[./PBMoltenSaltChannel]
  # Input parameters same as those in PBOneDFluidComponent are not listed.
  power_fraction          = (no_default)  # fraction of reactor power goes into
                                          # this molten salt channel
  power_product_name      = (no_default)  # scalar power product source names
  power_shape_function    = (no_default)  # axial power shape of the channel
  beta                    = (no_default)  # scalar power product fractions
  decay_curve_power_frac  = (no_default)  # Power fractions for decay heat curve
  decay_heat_curve_names  = (no_default)  # Decay heat curve names
[../]
```

- `power_fraction`

  This input parameter specifies the fraction of reactor power directly goes into this bypass channel. When specified, a `ReactorPower` component is also expected in the input file that provides the computation of reactor power. If not specified, it is assumed there is no direct heating to this PBMoltenSaltChannel.

- `power_shape_function`

  This input parameter accepts a function name, which can be a function of time and/or space. It is only needed when `power_fraction` is specified. It is important to note that the function value, which could be both temporal and spatial dependent, is multiplied to local volumetric heat source, and there is no re-normalization of total power. If not specified, SAM takes a default power shape function, which is defined as,

$$f = \frac{\pi}{2}\sin\left(\frac{\pi x}{L}\right)$$

  where x is the axial location, and L is the channel length.

- `power_product_name`

  This input parameter specifies a list of passive scalars that this PBMoltenSaltChannel component will model as its fission products. These passive scalars should have been defined in the global parameter, `passive_scalar` (see section 4.1). The product faction of each scalar is specified in `beta`. In addition this user-specified list of passive scalars, PBMoltenSaltChannel also has built-in decay curves for several isotopes, see `decay_heat_curve_names`.

- `beta`

  This parameter specifies the scalar power product fractions.

- `decay_heat_curve_names`

  This component also provides user input to choose built-in decay curves for several isotopes, including “U235T”, “PU239T”, “U238F”, “PU241T”, and “TEST235”.

4.3.8 FuelAssembly

`FuelAssembly` or `DuctedFuelAssembly` (see section 4.3.9) model the reactor fuel assemblies composed of multiple `PBCoreChannels` or `PBDuctedCoreChannels`, representing different axial regions of a fuel assembly including the active core, gas plenum, lower and upper reflector, lower and upper shield, etc. The junction components (`PBSingleJunction`) are also auto-created in `FuelAssembly` or

DuctedFuelAssembly to model the connection among the fluid parts of PBCoreChannel or PBDucted-CoreChannel.

The complete list of input parameters of FuelAssembly is give as,

```
[./FuelAssembly]
  A                         = (required)    # Areas of the OneDComp
  Dh                        = (required)    # Hydraulic diameter
  HTC_geometry_type         = Pipe          # Heat transfer geometry type
  HTC_user_option           = Default       # User option heat transfer correlations
  HT_surface_area_density = (required)    # Heating surface density
  Hw                        = (no_default) # Convective heat transfer coefficient
  PoD                       = (no_default) # pitch to diameter ratio for parallel bundle
                                            # heat transfer
  SC_HTC                    = (no_default) # Sensitivity coefficient for HTC, multiplicative
  Ts_init                   = (required)    # Initial solid temperature
  dim_hs                    = 2             # See PBCoreChannel
  elem_number_of_hs         = (required)    # Number of elements of each heat structure
  eos                       = (required)    # See PBCoreChannel
  f                         = (no_default) # friction
  hs_type                   = (no_default) # Geometry type of the fuel
  initial_P                 = (no_default) # Initial pressure in the OneDComp
  initial_T                 = (no_default) # Initial temperature in the OneDComp
  initial_V                 = (no_default) # Initial velocity in the OneDComp
  junction_nodalTbc         = 1             # if use nodalTbc for junctions
  lam_factor                = (no_default) # a user-input shape factor for laminar friction
                                            # factor for non-circular flow channels
  length                    = (required)    # Length of the OneDComp
  material_hs               = (required)    # Name of the materials used in the heat
                                            # structures
  n_assemblies              = 1             # See PBCoreChannel
  n_elems                   = (required)    # number of element in this OneDComp
  n_heatstruct              = (required)    # Number of heat structures
  n_zones                   = (required)    # number of zones
  name_of_hs                = (required)    # User given heat structure names
  orientation               = '0 0 1'       # See PBCoreChannel
  plate_depth               = (no_default) # The dimension of plate fuel in the third
                                            # direction, m
  position                  = '0 0 0'       # See PBCoreChannel
  power_fraction            = (no_default) # fraction of reactor power goes into this core
                                            # channel
  power_shape_function      = (no_default) # See PBCoreChannel
  rotation                  = 0             # See PBCoreChannel
  roughness                 = (no_default) # roughness, [m]
  scaling_velocity          = (no_default) # a user-input global velocity for PSPG scheme
  tao_pspg                  = (no_default) # tao_pspg
  tao_supg                  = (no_default) # tao_supg
  turb_factor               = (no_default) # a user-input shape factor for turbulent friction
                                            # factor for non-circular flow channels
  width_of_hs               = (required)    # Width of each heat structure
[../]
```

The input parameters of a FuelAssembly are quite similar to those of a PBCoreChannel. However, as a FuelAssembly consists of multiple PBCoreChannels to represent different axial regions of a fuel assembly, most of its input parameters require a list of numbers (or strings), instead of a single number (or string) as in PBCoreChannel. Most of these input parameters could be found in PBCoreChannel or PBOneDFluidComponent and HeatStructure.

- n_zones (required)

  This input parameter specifies number or zones, i.e., number of PBCoreChannels, to be used in the FuelAssembly component along the axial direction.

- `A` (required)

  A list of n_zones areas for the n_zones PBCoreChannels.

- `Dh` (required)

  A list of n_zones hydraulic diameters, D_h, for the n_zones PBCoreChannels.

- `n_elems` (required)

  A list of n_zones numbers of elements for the one-dimensional flow channel in the axial direction for the n_zones PBCoreChannels.

- `HTC_geometry_type`

  A list of n_zones heat transfer geometry types for the n_zones PBCoreChannels. For heat transfer geometry types, see section 4.3.1.

- `HTC_user_option`

  A list of n_zones heat transfer user options for the n_zones PBCoreChannels. For heat transfer user options, see section 4.3.1.

- `HT_surface_area_density` (required)

  A list of n_zones heat transfer surface densities, a_w, for the n_zones PBCoreChannels.

- `Hw`

  A list of n_zones wall heat transfer coefficients for the n_zones PBCoreChannels.

- `PoD`

  A list of n_zones pitch-to-diameter ratios for the n_zones PBCoreChannels.

- `SC_HTC`

  A list of n_zones sensitivity coefficients for heat transfer coefficient for the n_zones PBCoreChannels.

- `Ts_init` (required)

  A list of n_zones heat structure initial temperatures for the n_zones PBCoreChannels.

- `dim_hs`

  The dimension of the mesh used for the heat structure of all PBCoreChannels. It does not require a list of dimensions.

- `f`

  A list of n_zones user-specified wall frictional coefficient for the n_zones PBCoreChannels.

- `hs_type`

  A list of n_zones heat structure types for the n_zones PBCoreChannels.

- `initial_P`, `initial_T`, and `initial_V`

  A list of n_zones initial pressure (temperature, velocity) for the n_zones PBCoreChannels.

- `junction_nodalTbc`

  As within the FuelAssembly component, n_zones - 1 PBSingleJunctions are automatically created to connect the n_zones PBCoreChannels, this input parameter specifies if 'nodal_Tbc' is to be used for these automatically generated PBSingleJunctions. For 'nodal_Tbc' of PBSingleJunction, see section 4.3.22.

- `lam_factor` and `turb_factor`

  A list of n_zones user-input shape factors for laminar (turbulent) flow friction factor for non-circular flow channels for the n_zones PBCoreChannels.

- `length` (required)

  A list of n_zones lengths for the n_zones PBCoreChannels.

- `n_heatstruct` (required)

  A list of n_zones number of heat structures for the n_zones PBCoreChannels. Sum all these numbers up, the total number of heat structures, $n_{hs,total}$, is obtained. For example, in a two-zone FuelAssembly, if `n_heatstruct = '2 3'`, there are in total 5 heat structures ($n_{hs,total} = 5$) that will require use-inputs for their names, widths, materials, number of elements in its width direction, power fractions, etc. See following discussions.

- `elem_number_of_hs` (required)

  A list of $n_{hs,total}$ numbers of elements for the $n_{hs,total}$ heat structures in the FuelAssembly component.

- `material_hs` (required)

  A list of $n_{hs,total}$ names of materials for the $n_{hs,total}$ heat structures in the FuelAssembly component.

- `power_fraction` (required)

  A list of $n_{hs,total}$ power fractions that goes to the $n_{hs,total}$ heat structures in the FuelAssembly component.

- `plate_depth`

  A list of n_zones values of the dimension of plate fuel in the third direction for the n_zones PBCoreChannels. Only needed for plate type of fuels. Also, see section 4.3.4.

- `roughness`

  A list of n_zones wall roughnesses for the n_zones PBCoreChannels.

- `scaling_velocity`, `tao_pspg`, and `tao_supg`

  A list of n_zones scaling velocities (τ_{PSPG}, τ_{SUPG}) for the n_zones PBCoreChannels. Also, see section 4.3.4.

4.3.9 DuctedFuelAssembly

DuctedFuelAssembly is simply a FuelAssembly (see section 4.3.8) with an outer duct. It inherits all input parameters from FuelAssembly component, and requires additional input parameters to describe its outer duct:

```
[./DuctedFuelAssembly]
  # Input parameters same as those in FuelAssembly are not listed.

  dim_duct              = 2             # Dimension of the geometry (1 = 1D, 2 = 2D)
  Tduct_init            = (no_default)  # Initial duct wall temperature
  duct_thickness        = (required)    # Thickness of the duct wall
  elem_number_of_duct   = (required)    # number of elements in the duct wall
  material_duct         = (required)    # Name of the material used in the duct wall
  HT_surface_area_density_duct   = (required)  # duct side heating surface density

  input_parameters      = (no_default)  # Name of the ComponentInputParameters
                                        # user object
[../]
```

- `dim_duct`

  Similar to `dim_hs` of `HeatStructure`, it specifies how duct wall heat structure is modeled, either in one-dimensional (1) or two-dimensional (2). The default and recommended value is 2, i.e., two-dimensional.

- `Tduct_init`

  A list of n_zones initial temperatures for the duct wall of the n_zones PBDuctedCoreChannels. If not specified, it takes the global initial temperature, `global_init_T` (see section 4.1), as the initial duct wall temperature.

- `duct_thickness` (required)

  A list of n_zones duct wall thickness of the n_zones PBDuctedCoreChannels.

- `elem_number_of_duct` (required)

  A list of n_zones numbers of elements, each of which is to be used to specify the number of elements in the duct wall for each PBDuctedCoreChannel.

- `material_duct` (required)

  This parameter specifies the duct wall material for all duct walls.

- `HT_surface_area_density_duct` (required)

  A list of n_zones heat transfer surface area densities of the duct walls with respect to the PBDuctedCoreChannels.

- `input_parameters`

  See section 4.5.

4.3.10 MultiChannelRodBundle

To improve the heat transfer between the duct wall and coolant flow, a multi-channel rod bundle model is developed in SAM to account for the temperature differences between the center region and the edge region of the coolant channel in a fuel assembly. Similar approach has been proposed in ENERGY (Yang and Joo, 1999), SAS4A/SASSYS-1 (Fanning 2012), as well as the multi-region porous medium model reported by Yu et al. (2015). The whole fuel assembly can be divided into a number of regions, as shown in figure 4.9. It is quite remarkable that the volumetric heat flux in region 1 is significantly less than that in other regions, based on analytical calculations. Each inner region is modeled as an average core-channel (i.e., a 1-D coolant channel and an average fuel pin). The edge region can be modeled as one core-channel or six core-channels to account for the differences in heat transfer with each side of the duct wall. This zoning strategy is also inspired from the authors' previous experiences in the CFD simulations of the triangle-lattice pin bundles. As shown in the Hu and Yu (2016), large temperature gradient were observed in the coolant region near the duct wall, but the temperature distribution elsewhere is very uniform except the hot spots due to the wire-wrap effects.

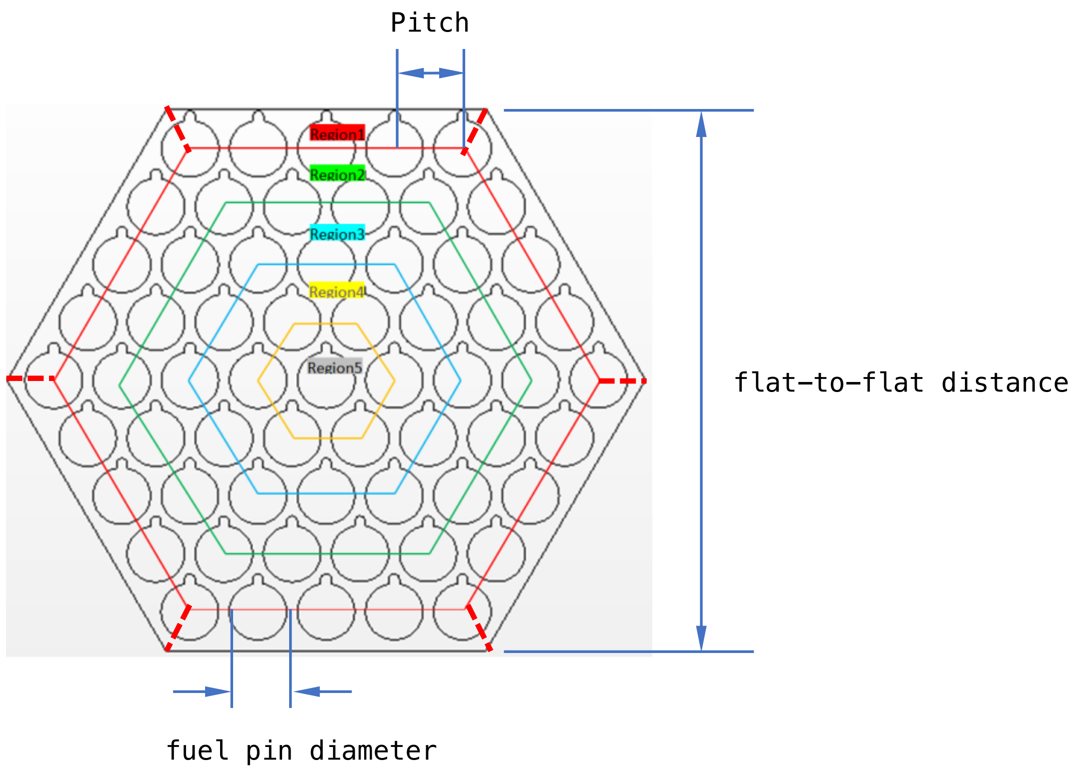


Figure 4.9: Sketch of the regions in the multi-channel model.

In the SAM multi-channel model, the fluid regions are modeled as separate parallel channels with the same pressure drop. For simplicity, it is assumed that there are no mass and momentum exchange between channels. However, the energy exchange is allowed at all axial nodes.

MultiChannelRodBundle assumes that the heat structures are the same for all its automatically generated PBCoreChannels, and thus the same input parameters for heat structure are used for all PBCoreChannels. Also, note that MultiChannelRodBundle does not model the outer duct wall.

Input parameters of MultiChannelRodBundle are given as follows:

```
[./MultiChannelRodBundle]
  Ts_init               = (required)        # See PBCoreChannels, same for all channels
  assem_Dft             = (required)        # The flat-to-flat distance of the assembly
  beta                  = 0                 # turbulent mixing parameter
  dim_hs                = 1                 # The dimension of the mesh used for the
                                            # heat structure: 1 = 1D(default), 2 = 2D .
  elem_number_of_hs     = (required)        # See PBCoreChannels, same for all channels
  eos                   = (required)        # See PBOneDFluidComponent
  initial_P             = (no_default)      # Initial pressure in the OneDComp
  initial_T             = (no_default)      # Initial temperature in the OneDComp
  initial_V             = (no_default)      # Initial velocity in the OneDComp
  lam_factor            = (no_default)      # a user-input shape factor for laminar
                                            # friction factor for non-circular flow channels
  length                = (required)        # See PBCoreChannel, same for all channels
  material_hs           = (required)        # See PBCoreChannel, same for all channels
  n_channel             = (no_default)      # Number of CoreChannels
  n_elems               = (required)        # number of axial element
  n_heatstruct          = (required)        # See PBCoreChannels, same for all channels
  n_side                = (required)        # Number of fuel pin rings
  n_zones               = (required)        # Number of zones
  name_of_hs            = (required)        # See PBCoreChannels, same for all channels
  orientation           = '0 0 1'           # See PBOneDFluidComponent
  pin_diameter          = (required)        # The fuel pin diameter
  pin_pitch             = (required)        # The distance between fuel pin centers
  position              = '0 0 0'           # See PBOneDFluidComponent
  power_fraction        = (no_default)      # See PBCoreChannel, same for all channels
  power_shape_function = (no_default)       # See PBCoreChannel, same for all channels
  radial_power_peaking = (no_default)       # radial power peaking factors
  rotation              = 0                 # See PBOneDFluidComponent
  roughness             = (no_default)      # roughness, [m]
  scaling_velocity      = (no_default)      # a user-input global velocity for PSPG scheme
  tao_pspg              = (no_default)      # tao_pspg
  tao_supg              = (no_default)      # tao_supg
  turb_factor           = (no_default)      # a user-input shape factor for turbulent
                                            # friction factor for non-circular flow channels
  width_of_hs           = (required)        # See PBCoreChannels, same for all channels
  wire_diameter         = 0                 # The wire wrap diameter

  input_parameters      = (no_default)      # Name of the ComponentInputParameters
                                            # user object
[../]
```

- `n_zones (required)`

  Number of zones (regions) of the MultiChannelRodBundle. For example, `n_zones = 5` in figure 4.9. Note that it is different than the same parameter defined in FuelAssembly (see section 4.3.8), which splits the fuel assembly in the axial direction.

- `n_side (required)`

  Number of fuel pin rings of the MultiChannelRodBundle. As shown in figure 4.9, it is also the same as the number of fuel pins on the side of MultiChannelRodBundle. In figure 4.9, `n_side = 5`.

- `n_channel (required)`

Number of core-channels to be modeled in this MultiChannelRodBundle component. It has to be either the same as `n_zones`, such that one core-channel for each zone, or equal to `n_zones+5`, such that the out most zone is modeled as six (6) core-channels and one core-channel for each inner zone. For example, as shown in figure 4.9, if `n_channel = 5`, one core-channel will be created for each zone (region) 1 to 5; however, if `n_channel = 10`, one core-channel will be created for each zone (region) 2 to 5, while 6 core-channels are to be created for zone (region) 1. One can easily figure out that MultiChannelRodBundle currently only supports a hexagonal fuel bundles such as the one shown in figure 4.9.

- `assem_Dft (required)`

  This parameter specifies the flat-to-flat distance of the assembly, see figure 4.9.

- `beta`

  Turbulent mixing parameter which will be used to compute the turbulent mixing between neighboring core-channels. Currently, it accepts a simple number for all mixing parameters. This will be improved in future implementations.

- `dim_hs`

  Same for all core-channels of this MultiChannelRodBundle, it specifies how the heat structure is modeled, either in one-dimensional (`dim_hs` = 1) or two-dimensional (`dim_hs` = 2). The default and recommended value is 1, i.e., one-dimensional.

- `initial_P`, `initial_T`, and `initial_V`

  A list of n_zones initial pressure (,temperature, and velocity) for the n_zones PBCoreChannels.

- `lam_factor` and `turb_factor`

  A list of n_zones user-input shape factors for laminar (turbulent) flow friction factor for non-circular flow channels for the n_zones PBCoreChannels.

- `roughness`

  A list of n_zones wall roughnesses for the n_zones PBCoreChannels.

- `scaling_velocity`, `tao_pspg`, and `tao_supg`

  A list of n_zones scaling velocities (,τ_{PSPG}, and τ_{SUPG}) for the n_zones PBCoreChannels. Also, see section 4.3.4.

- `wire_diameter`

  The wire wrap diameter, same for all PBCoreChannels. The default value is zero.

- `pin_diameter (required)`

  The fuel pin diameter, same for all PBCoreChannels, see figure 4.9.

- `pin_pitch (required)`

  The distance between fuel pin centers, see figure 4.9.

- `input_parameters`

  See section 4.5.

4.3.11 HexLatticeCore

HexLatticeCore models a reactor core with a hexagonal lattice such as SFRs, as schematically shown in figure 4.10. It can automatically generate the core lattice of MultiChannelRodBundle or PBCoreChannel components, and the inter-assembly structures (including duct walls and inter-assembly gaps), based on the geometry information specified in the input.

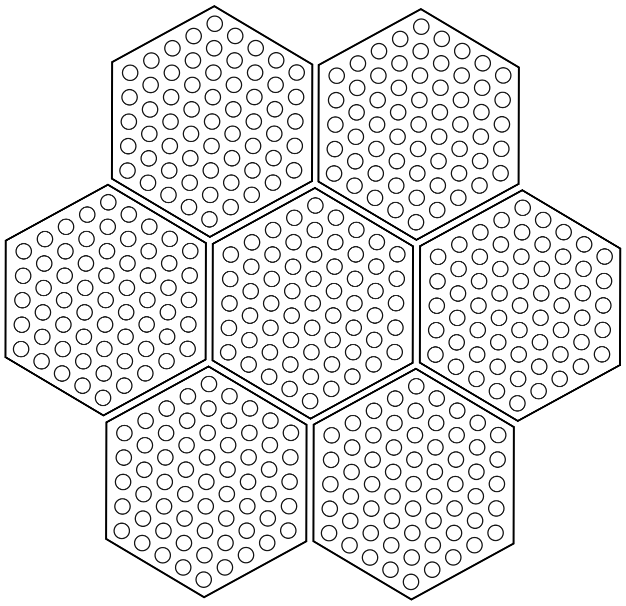


Figure 4.10: Sketch of HexLatticeCore component.

Its input parameters are given as follows:

```
[./HexLatticeCore]
  assem_Dft              = (required)     # The flat-to-flat distance of the assembly
  assem_layout           = (required)     # The layout of the assembly lattice
  assem_pitch            = (required)     # The distance between assembly centers
  b_multichannel         = 0              # if use MultiChannelRodBundle for each assembly
  b_radial_heat_transfer  = 1              # if modeling radial heat transfer between
                                          # assemblies
  n_side                 = (required)     # Number of side CoreChannels
  orientation            = '0 0 1'        # See PBOneDFluidComponent
  position               = '0 0 0'        # See PBOneDFluidComponent
  radial_power_peaking   = (no_default)   # radial power peaking factors
  ref_duct               = (no_default)   # reference heat structure for one side of an
                                          # assembly duct wall
  ref_hs                 = (no_default)   # reference heat structure for intra-assembly gap
                                          # and two duct walls
  rotation               = 0              # See PBOneDFluidComponent
[../]
```

- n_side (required)

  Number of assemblies on each side of the core. For example, n_side = 2 in figure 4.10, and the total number of assemblies in the entire core is 7.

- assem_Dft (required)

The flat-to-flat distance of the assembly, assumed to be the same for all assemblies in the HexLatticeCore.

- `assem_pitch` (required)

  The distance between assembly centers, assumed to be the same for all assemblies in the HexLatticeCore.

- `b_multichannel`

  This input parameter specifies if MultiChannelRodBundle (true) or simply PBCoreChannel (false) should be used to represent each assembly in the core. By default, it is 'false', i.e., to use PBCoreChannel.

- `assem_layout` (required)

  This input parameter accepts a list of names for ComponentInputParameters, which are to be used to automatically create all assemblies within the HexLatticeCore component. The number of ComponentInputParameters has to be the same as the number of assemblies in the core, and the assemblies are numbered in an order from left to right, and then top to bottom. If `b_multichannel = true`, it is expected that the list of ComponentInputParameters are of `MultiChannelRodBundleParameters` type, otherwise, `PBCoreChannelParameters`.

- `b_radial_heat_transfer`

  This input parameter specifies if radial heat transfer between assemblies should be modeled. If true (default), HexLatticeCore component automatically adds duct wall heat structures between assemblies, and to the out most boundaries of all assemblies, and models the heat transfer between duct walls/gaps and their neighboring core channels.

- `ref_hs`

  If radial heat transfer between assemblies should be modeled, this input parameter accepts a `HeatStructureParameters` to build duct wall heat structures between two neighboring assemblies. It is recommended that this reference heat structure models three layers of heat structures, including two duct walls and a layer of gap (modeled as a layer heat structure) between them. It assumes that all such intra-assemblies heat structures have similar geometries.

- `ref_duct`

  If radial heat transfer between assemblies should be modeled, this input parameter accepts a `HeatStructureParameters` to build duct wall heat structures for the out most boundaries of all assemblies. This reference heat structure should contain only one layer of duct wall. It assumes that all duct wall heat structures have similar geometries.

- `radial_power_peaking`

  This input parameter accepts a list of radial power peaking factors for assemblies within the HexLatticeCore component. The number of power peaking factors has to be the same as the total number of assemblies. The power fraction that goes to each assembly is simply calculated as power peaking factor divided by total number of assemblies. If not specified, it assumes that power is uniformly distributed across all assemblies.

4.3.12 PBCoupledHeatStructure

PBCoupledHeatStructure simulates a HeatStructure with controlled boundary conditions at the two surfaces, such as adiabatic, fixed temperature, convective heat transfer with ambient, or coupled with 0-D liquid volume or 1-D liquid components. Normally users will not directly use HeatStructures to create their models, but use PBCoupledHeatStructure instead.

Most its input parameters are inherited from the base HeatStructure component, see section 4.3.2, and additional input parameters are required to setup its left(right) boundary conditions. The complete input parameters of PBCoupledHeatStructure are listed below:

```
[./PBCoupledHeatStructure]
  Ts_init               = (no_default)    # See HeatStructure
  axial_offset          = 0               # See HeatStructure
  depth_plate           = (no_default)    # See HeatStructure
  dim_hs                = 2               # See HeatStructure
  elem_number_axial     = 1               # See HeatStructure
  elem_number_radial    = (required)      # See HeatStructure
  end_elems_refinement  = 1               # See HeatStructure
  hs_names              = (no_default)    # See HeatStructure
  hs_power              = (no_default)    # See HeatStructure
  hs_power_shape_fn     = (no_default)    # See HeatStructure
  hs_type               = plate           # See HeatStructure
  heat_source_solid     = 0               # See HeatStructure
  input_parameters      = (no_default)    # See HeatStructure
  length                = (required)      # See HeatStructure
  material_hs           = (required)      # See HeatStructure
  offset                = '0 0 0'         # See HeatStructure
  orientation           = '0 0 1'         # See HeatStructure
  position              = '0 0 0'         # See HeatStructure
  power_fraction        = (no_default)    # See HeatStructure
  radius_i              = (no_default)    # See HeatStructure
  rotation              = 0               # See HeatStructure
  width_of_hs           = (required)      # See HeatStructure

  D_heated_left         = (no_default)  # Characteristic heated length at left surface
  D_heated_right        = (no_default)  # Characteristic heated length  at right surface
  HS_BC_type            = (required)    # Heat structure boundary condition type.
                                        # Allowed options (case sensitive):
                                        # Temperature; Convective; Coupled
  Hw_left               = (no_default)  # Convective heat transfer coefficient at
                                        # left surface
  Hw_right              = (no_default)  # Convective heat transfer coefficient at
                                        # right surface
  T_amb_left            = 300           # left ambient temperature
  T_amb_right           = 300           # right ambient temperature
  T_bc_left             = 600           # Fixed Temperature BC at left surface
  T_bc_right            = 600           # Fixed Temperature BC
  T_external_left       = (no_default)  # Coupled variable for left external temperature
  T_external_right      = (no_default)  # Coupled variable for right external temperature
  h_external_left       = (no_default)  # Coupled variable for left external heat
                                        # transfer coefficients
  h_external_right      = (no_default)  # Coupled variable for right external heat
                                        # transfer coefficients
  name_comp_left        = (no_default)  # The name of the left liquid volume connected
                                        # to the heat structure
  name_comp_right       = (no_default)  # The name of the right liquid volume connected
                                        # to the heat structure
  qs_external_left      = (no_default)  # Coupled variable for left heat flux
  qs_external_right     = (no_default)  # Coupled variable for right heat flux
  qs_left               = (no_default)  # Heat flux at the left surface
  qs_right              = (no_default)  # Heat flux at the right surface
  HTC_geometry_type_left         = Pipe # Heat transfer geometry type at left surface
  HTC_geometry_type_right        = Pipe # Heat transfer geometry type at left surface
  heat_transfer_area_left        = 1    # Convective heat transfer area at left surface
  heat_transfer_area_right       = 1    # Convective heat transfer area at right surface
  HT_surface_area_density_left   = 1  # Heat transfer surface area density at left surface
  HT_surface_area_density_right  = 1  # Heat transfer surface area density at right surface
  HT_area_multiplier_left        = 1  # HT surface area density multiplier at left surface
  HT_area_multiplier_right       = 1  # HT surface area density multiplier at right surface
[../]
```

- `HS_BC_type` (required)

  This input parameter specifies the two boundary condition types for the left and right side of PBCoupledHeatStructure component. Input options include “Temperature”, “Convective”, and “Coupled”.

  If “Temperature” is specified, a Dirichlet type of boundary conditions is used, and it expects an additional user-input for the boundary condition temperature, see `T_bc_left(right)`.

  If “Convective” is specified, a wall heat flux will be used. The wall heat flux could be directly specified, see `qs_left(right)`. It is also possible to compute the wall surface flux by providing ambient temperature (see `T_amb_left(right)`) and heat transfer coefficient to the ambient (see `Hw_left(right)`).

  In code coupling computation situations, wall heat flux boundary conditions could be calculated from an external code. Corresponding to the two conditions discussed above, one could specify an externally calculated wall heat flux, see `qs_external_left(right)`; or a combination of external temperature and wall heat transfer coefficient, see `T_external_left(right)` and `h_external_left(right)`.

  If “Coupled” is specified, still a wall heat flux will be used, but the fluid temperature comes from a neighboring component, either a PBVolumeBranch or a PBOneDFluidComponent, see `name_comp_left(right)`. Heat transfer coefficient to the coupled component can be specified in `Hw_left(right)`, and if not specified, they will be automatically computed.

- `T_bc_left` and `T_bc_right`

  If the left/right boundary condition type is “Temperature”, it accepts the value (or a function) for the left/right boundary condition temperatures. The default values are 600 K.

- `qs_left` and `qs_right`

  If the left/right boundary condition type is “Convective”, it accepts the value (or a function) for the left/right wall heat flux as boundary conditions.

- `T_amb_left` and `T_amb_right`; `Hw_left` and `Hw_right`

  If the left/right boundary condition type is “Convective”, `T_amb_left(right)`) specifies the ambient temperature, and `Hw_left(right)`) specifies the wall heat transfer coefficient to compute the wall heat flux. Both of these inputs could be either values or function names. The default value for `T_amb_left(right)` is 300 K.

- `qs_external_left` and `qs_external_right`

  If the left/right boundary condition type is “Convective”, it accepts the variable name for externally computed left/right wall heat flux as boundary conditions.

- `T_external_left` and `T_external_right`; `h_external_left` and `h_external_right`

  If the left/right boundary condition type is "Convective", `T_external_left(right)` specifies an externally computed ambient temperature, and `h_external_left(right)` specifies an externally computed wall heat transfer coefficient to compute the wall heat flux. Both of these inputs have to be specified as coupled "variables".

- `name_comp_left` and `name_comp_right`

  If the left/right boundary condition type is "Coupled", it accepts the name of the component coupled to the left(right) surface of this PBCoupledHeatStructure component. The coupled component has to be either a PBVolumeBranch or a PBOneDFluidComponent.

- `D_heated_left` and `D_heated_right`

  Only when the left/right boundary condition type is "Coupled", and the coupled component is of PBOneDFluidComponent type, it specifies the characteristic length to compute wall heat transfer coefficient in the coupled left(right) PBOneDFluidComponent component. If not specified, the coupled PBOneDFluidComponent uses its hydraulic diameter as the characteristic length for heat transfer.

- `HTC_geometry_type_left` and `HTC_geometry_type_right`

  Heat transfer geometry type at the left(right) surface. Acceptable options are "Pipe (default)", "Bundle", "Vertical-Plate", and "Horizontal-Plate".

- `HT_surface_area_density_left` and `HT_surface_area_density_right`

  Only when the left/right boundary condition type is "Coupled", and the coupled component is of PBOneDFluidComponent type, it specifies the heat transfer surface area density of this PBCoupledHeatStructure component with respect to the coupled left(right) PBOneDFluidComponent component.

- `heat_transfer_area_left` and `heat_transfer_area_right`

  Only when the left/right boundary condition type is "Coupled", and the coupled component is of PBVolumeBranch type, it specifies the left(right) side heat transfer surface area of this PBCoupledHeatStructure component.

- `HT_area_multiplier_left` and `HT_area_multiplier_right`

  In cases where complex heat structures have to be simplified to simpler geometries (e.g., cylinder), or heat structure surfaces are only partially heated/cooled by the coupled flow component, left (right) surface area multiplier `HT_area_multiplier_left(right)` is used to correct such a distortion effect.

  In any case, the coupled left (right) flow component recieves an energy source as $ha_w\Delta T$, in which a_w is the left (right) heat surface area density based on true geometry of the complex heat structure geometry, see `HT_surface_area_density_left(right)`, h the heat transfer coefficient, and ΔT the difference between heat strucutre surface temperature and fluid temperature. Because of geometry simplification, the simplified heat strucutre surface area is no longer the same as the true surface area, and therefore a multiplier is used to correct such a distortion. The multiplier value should be set as the ratio of true heat transfer area to the simplified heat strucutre surface area.

4.3.13 HeatStructureWithExternalFlow

HeatStructureWithExternalFlow is also a HeatStructure-based component similar to PBCoupledHeatStructure, however with the main purpose to facilitate code-to-code coupling via its boundary surfaces, either using MOOSE's MultiApp infrastructure or using data exchange with non-MOOSE-based application. When using MOOSE's MultiApp infrastructure, this external MOOSE simulation (a.k.a., MOOSE subApp) will need a different component, HeatTransferWithExternalHeatStructure (see section 4.3.14) to realize data exchange. It is always assumed that its left-side surface is coupled to an external code, while its right-side surface is reserved for SAM to handle its boundary condition.

```
[./HeatStructureWithExternalFlow]
  Ts_init                   = (no_default)       # See HeatStructure, NOT USED.
  axial_offset              = 0                  # See HeatStructure
  depth_plate               = (no_default)       # See HeatStructure
  dim_hs                    = 2                  # See HeatStructure
  elem_number_axial         = 1                  # See HeatStructure
  elem_number_radial        = (required)         # See HeatStructure
  end_elems_refinement      = 1                  # See HeatStructure
  hs_names                  = (no_default)       # See HeatStructure
  hs_power                  = (no_default)       # See HeatStructure
  hs_power_shape_fn         = (no_default)       # See HeatStructure
  hs_type                   = plate              # See HeatStructure
  length                    = (required)         # See HeatStructure
  material_hs               = (required)         # See HeatStructure
  offset                    = '0 0 0'            # See HeatStructure
  orientation               = '0 0 1'            # See HeatStructure
  position                  = '0 0 0'            # See HeatStructure
  power_fraction            = (no_default)       # See HeatStructure
  radius_i                  = (no_default)       # See HeatStructure
  rotation                  = 0                  # See HeatStructure
  width_of_hs               = (required)         # See HeatStructure
  heat_source_solid        = 0                   # See HeatStructure

  HS_BC_type                = (required)    # Heat structure boundary condition type
  Hw_internal               = None          # convective heat transfer coefficient
                                            # at SAM Side surface
  T_bc_internal             = 600           # Fixed Temperature BC
  T_external_init           = 600           # Initial heat structure temperature
  T_identifier_in_file      = (no_default)  # External temperature identifier
                                            # used in external data file
  T_internal                = 300           # Sam Side ambient temperature
  delete_data_file          = (no_default)  # Delete data file after reading
  eos_internal              = None          # The name of EOS to use
  h_external_init           = 3000          # Initial heat transfer coefficient
  h_identifier_in_file      = (no_default)  # External heat transfer coefficient
                                            # identifier used in external data file
  input_data_file           = (no_default)  # The file name used to save external
                                            # coupling information
  name_comp_internal        = None          # The name of the Sam Side liquid
                                            # volume connected to the heat structure
  output_data_file          = (no_default)  # The file name used to save SAM
                                            # output information for coupling
  output_mesh               = (no_default)  # The mesh used to output SAM data
  output_template_file      = (no_default)  # The template file name used to
                                            # save SAM output information for coupling
  qs_external_init          = 0             # Initial heat flux
  qs_internal               = None          # Heat flux at the SAM Side surface

  HT_surface_area_density_internal = 1      # heat transfer surface area density
                                            # at Sam Side surface
  heat_transfer_area_internal       = 1     # convective heat transfer area
                                            # at Sam Side surface
[../]
```

- HS_BC_type (required)

  This input parameter specifies: 1) a list of two heat structure boundary condition types for the left-side (coupled to an external code) and right-side (within SAM) of this component; or 2) a list of one heat structure boundary condition type just for the left-side (coupled to an external code) of this component, while the right-side surface assumes a zero heat flux boundary condition. For the left-side boundary condition type, it could be one of "CoupledConvection_T_h", "CoupledConvection_q", or "PpsConvective".

  Using MOOSE's MultiApp infrastructure, "CoupledConvection_T_h" facilitates a surface heat flux coupling via externally computed temperature and heat transfer coefficient, which also requires additional input parameters, such as:
  T_external_init
  h_external_init.

  Using MOOSE's MultiApp infrastructure, "CoupledConvection_q" simply takes an externally computed wall heat flux as its boundary condition. It requires an additional input parameter, qs_external_init.

  "PpsConvective" facilitates SAM code coupling with a non-MOOSE-based code, such as SAS. The coupling mechanism is similar to "CoupledConvection_T_h", which also computes its wall heat flux from externally computed fluid temperature and heat transfer coefficient. Additional input parameters are required for these external fluid temperature (see T_identifier_in_file) and heat transfer coefficient (see h_identifier_in_file).

  The right-side boundary condition type of this heat structure can be "Temperature", "Convective", or "Coupled", the same as defined in PBCoupledHeatStructure component, see section 4.3.12.

- T_external_init

  This input parameter specifies the initial value for the external temperature variable. The default value is 600 K.

- h_external_init

  This input parameter specifies the initial value for the external heat transfer coefficient variable. The default value is 3000 W/m^{2}K.

- qs_external_init

  This input parameter specifies the initial value for the externally computed wall heat flux variable. The default value is 0.

- input_data_file, T_identifier_in_file, and h_identifier_in_file

  When "PpsConvective" is used, input_data_file specifies the file name, where the external fluid temperature and heat transfer coefficient are stored. T_identifier_in_file specifies the name of external fluid temperature stored in this file. h_identifier_in_file specifies the name of external heat transfer coefficient stored in this file.

- `delete_data_file`

  This input parameter specifies if the external input file (see `input_data_file`) should be deleted after data being extracted from it.

- `output_data_file`, `output_mesh`, `output_template_file`

  When coupled to an external code, such as SAS, these input parameters specify the output file name to store SAM's out-going data, the mesh to be used for SAM's out-going data, and the template file that this out-going data file should follow.

- `T_bc_internal`

  If the right-side surface boundary condition type is "Temperature", this input parameter specifies the surface temperature value (or a function name).

- `T_internal` and `Hw_internal`

  If the right-side surface boundary condition type is "Convective", as one of two available options, this input parameter specifies the fluid temperature and heat transfer coefficient values. The other option is to specify `qs_internal`.

- `qs_internal`

  If the right-side surface boundary condition type is "Convective", this input parameter specifies the surface flux value (or a function name).

- `name_comp_internal` and `eos_internal`

  If the right-side surface boundary condition type is "Coupled", these input parameters specify the name of, and equation of state used in this coupled component.

- `HT_surface_area_density_internal`

  If the right-side surface boundary condition type is "Convective", and the coupled component is of one-dimensional flow type, such as PBOneDFluidComponent and PBCoreChannel, this input parameter specifies the heat transfer area density of this heat structure. The default value is 1 [m^2/m^3] (or [1/m]).

- `heat_transfer_area_internal`

  If the right-side surface boundary condition type is "Convective", and the coupled component is of zero-dimensional flow type, such as PBVolumeBranch, this input parameter specifies the heat transfer area of this heat structure. The default value is 1 [m^2].

4.3.14 HeatTransferWithExternalHeatStructure

This is a non-geometric type of component to facilitate data exchange between two MOOSE-based simulations via its MultiApp infrastructure. This component is intended to take the wall temperature from an external MOOSE-based application (i.e., a master mooseApp), while export its fluid component's fluid temperature and heat transfer coefficient to this external application, in order to compute the conjugate heat transfer.

```
[./HeatTransferWithExternalHeatStructure]
  T_wall_name       = T_wall_external      # Wall temperature variable name
  elemental_vars    = 0                    # if use elemental variables for T_wall and htc
  flow_component    = (required)           # Name of the flow component
  htc_name          = htc_external         # Heat transfer coefficient variable name
  initial_T_wall    = (required)           # External app wall temperature IC value
[../]
```

- `flow_component` (required)

  The name of the flow component, to which an external HeatStructure is coupled with, i.e., where the external heat flux to be applied.

- `T_wall_name`

  The name of incoming external wall temperature name, which will be transferred from an external MooseApp. Their default values are "T_wall_external".

- `initial_T_wall` (required)

  This input parameter specifies the initial value for the externally computed wall temperature before data transfer begins.

- `htc_name`

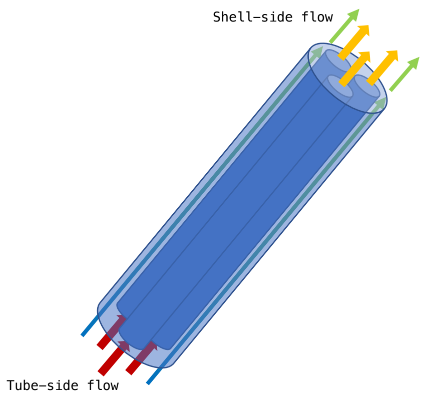
  The name of outgoing heat transfer coefficient name, which is computed in the flow component (see `flow_component`) and to be transferred to the external MooseApp. Their default values are "T_wall_external".

- `elemental_vars`

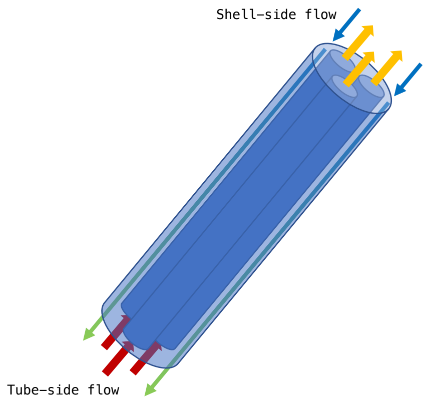
  This input parameter specifies if elemental type of variables, in contrast to nodal type of variable, are to be used for the outgoing heat transfer coefficient and the incoming external wall temperature.

4.3.15 PBHeatExchanger

PBHeatExchanger simulates a heat exchanger, including the fluid flow in the primary and secondary sides, convective heat transfer, and the heat conduction in the tube wall. Both counter-current and concurrent heat exchangers can be modeled. The two sides of the heat exchanger can have different orientation, lengths, flow areas, and hydraulic diameters. This gives the users more flexibilities to model a generic heat exchanger, including advanced heat exchangers being pursued by advanced reactor designs. Note that the two fluid sides of the heat exchanger and the tube wall must have the same number of elements axially.



(a) Concurrent design



(b) Counter-current design

Figure 4.11: Two types of PBHeatExchanger component designs. As an example, the two figures show shell-and-tube heat exchanger design.

The input parameters of PBHeatExchanger include those to setup the primary and secondary flow pipes, as well as those to setup the heat structure between the two pipes. Most of them are referred to PBOneDFluidComponent (see section 4.3.1) or HeatStructure (see section 4.3.2).

```
[./PBHeatExchanger]
  HX_type                                = Countercurrent        # Heat exchanger type

  A                                      = (required)            # See PBOneDFluidComponent
  A_secondary                            = (required)            # See PBOneDFluidComponent
                                                                 # (secondary side)
  Dh                                     = (required)            # See PBOneDFluidComponent
  Dh_secondary                           = (required)            # See PBOneDFluidComponent
                                                                 # (secondary side)
  HTC_geometry_type                      = Pipe                  # See PBOneDFluidComponent
  HTC_geometry_type_secondary            = Pipe                  # See PBOneDFluidComponent
                                                                 # (secondary side)
  HTC_user_option                        = Default               # See PBOneDFluidComponent
  HTC_user_option_secondary              = Default               # See PBOneDFluidComponent
                                                                 # (secondary side)
  HT_surface_area_density                = (no_default)          # See PBOneDFluidComponent
  HT_surface_area_density_secondary      = (required)            # See PBOneDFluidComponent
                                                                 # (secondary side)
  HoD                                    = 1                     # See PBOneDFluidComponent
  Hw                                     = (no_default)          # See PBOneDFluidComponent
  Hw_secondary                           = (no_default)          # See PBOneDFluidComponent
                                                                 # (secondary side)
  Ph                                     = (no_default)          # See PBOneDFluidComponent
  PoD                                    = 1                     # See PBOneDFluidComponent
  PoD_secondary                          = 1                     # See PBOneDFluidComponent
                                                                 # (secondary side)
  SC_HTC                                 = 1                     # See PBOneDFluidComponent
  SC_HTC_secondary                       = (no_default)          # See PBOneDFluidComponent
                                                                 # (secondary side)
  SC_WF                                  = 1                     # See PBOneDFluidComponent
  SC_WF_secondary                        = (no_default)          # See PBOneDFluidComponent
                                                                 # (secondary side)
  Twall_init                             = (required)        # See Ts_init of HeatStructure
  User_defined_HTC_parameters            = '0 0 0 0 0 0 0'       # See PBOneDFluidComponent
  User_defined_HTC_parameters_secondary  = '0 0 0 0 0 0 0'       # See PBOneDFluidComponent
                                                                 # (secondary side)
```

```
User_defined_WF_parameters            = '0 0 0'              # See PBOneDFluidComponent
User_defined_WF_parameters_secondary  = '0 0 0'              # See PBOneDFluidComponent
                                                             # (secondary side)
WF_geometry_type                      = Pipe                 # See PBOneDFluidComponent
WF_geometry_type_secondary            = Pipe                 # See PBOneDFluidComponent
                                                             # (secondary side)
WF_user_option                        = Default              # See PBOneDFluidComponent
WF_user_option_secondary              = Default              # See PBOneDFluidComponent
                                                             # (secondary side)
dim_wall                              = 2                 # See dim_hs of HeatStructure
end_elems_refinement                  = 1       # See both PBOneDFluidComponent
                                                # (for both primary and secondary side)
                                                # and HeatStructure
eos                                   = (required)           # See PBOneDFluidComponent
eos_secondary                         = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
f                                     = (no_default)         # See PBOneDFluidComponent
f_secondary                           = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
fluid_conduction         = (no_default)         # if modeling the fluid axial conduction
heat_source                           = 0.                   # See PBOneDFluidComponent
heat_source_secondary                 = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
hs_type                               = plate             # See dim_hs of HeatStructure
initial_P                             = (no_default)         # See PBOneDFluidComponent
initial_PS                            = (no_default)         # See PBOneDFluidComponent
initial_P_secondary                   = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
initial_T                             = (no_default)         # See PBOneDFluidComponent
initial_T_secondary                   = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
initial_V                             = (no_default)         # See PBOneDFluidComponent
initial_V_secondary                   = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
lam_factor                            = 1                    # See PBOneDFluidComponent
lam_factor_secondary                  = 1                    # See PBOneDFluidComponent
                                                             # (secondary side)
length                                = (required)           # See PBOneDFluidComponent
length_secondary                      = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
material_wall           = (required)           # See material_hs of HeatStructure
n_elems                               = (required)           # See PBOneDFluidComponent
n_wall_elems            = (required)           # See elem_number_radial of HeatStructure
orientation                           = '0 0 1'              # See PBOneDFluidComponent
orientation_secondary                 = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
position                              = '0 0 0'              # See PBOneDFluidComponent
radius_i                              = (no_default)         # See HeatStructure
rotation                              = 0                    # See PBOneDFluidComponent
roughness                             = 0                    # See PBOneDFluidComponent
roughness_secondary                   = 0                    # See PBOneDFluidComponent
                                                             # (secondary side)
scalar_source                         = (no_default)         # See PBOneDFluidComponent
scalar_source_secondary               = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
scaling_velocity                      = (no_default)         # See PBOneDFluidComponent
scaling_velocity_secondary            = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
tao_pspg                              = (no_default)         # See PBOneDFluidComponent
tao_pspg_secondary                    = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
tao_supg                              = (no_default)         # See PBOneDFluidComponent
tao_supg_secondary                    = (no_default)         # See PBOneDFluidComponent
                                                             # (secondary side)
```

```
  turb_factor                          = 1                   # See PBOneDFluidComponent
  turb_factor_secondary                = 1                   # See PBOneDFluidComponent
                                                             # (secondary side)
  wall_thickness                       = (required)   # See width_of_hs of HeatStructure

  heat_transfer_area_error_tolerance   = 0.001    # The ratio of (length*Phf) at two
                                                  # sides of HX must be equal to 1 for
                                                  # plate type and be equal to the ratio
                                                  # of inner and outer pipe diameters
                                                  # for cylinder type within this relative
                                                  # tolerance value
[../]
```

- HX_type

  This input parameter specifies the type of heat exchanger, either "Concurrent" or "Countercurrent (default)". The two types of heat exchanger designs are shown in figure 4.11.

- heat_transfer_area_error_tolerance

  This input parameter specifies an error tolerance, if the user-input parameters cause wall heat structure two-side surface areas inconsistence (see discussion below). The default value is 10^{-3}.

  Since the heat exchanger input parameters are organized in a way that the they are provided separately for three sub-components, i.e., the primary-side pipe, the secondary-side pipe, and the solid wall between them, user-input inconsistence could often raise, which could cause inconsistence in heat transfer areas between the two sides, and consequently introduces error in overall heat exchanger energy balance.

  If the solid-wall heat structure is of "Plate" type, the heat transfer areas on both sides are the same, which requires,

$$A_p a_{w,p} L_p = A_s a_{w,s} L_s;$$

  and for "cylinder", the two-side areas follows,

$$\frac{A_p a_{w,p} L_p}{A_s a_{w,s} L_s} = \frac{r_p}{r_s};$$

  in which, A is flow area, a_w is heat transfer area density, L is pipe length, r is the radius on wall surface; subscripts p and s denote the primary- and secondary-side, respectively.

  The input heat transfer area error is then defined as,

$$\varepsilon = \left| \frac{A_p a_{w,p} L_p}{A_s a_{w,s} L_s} - 1 \right|$$

  for plate type of wall heat structure, and

$$\varepsilon = \left| \frac{A_p a_{w,p} L_p}{A_s a_{w,s} L_s} - \frac{r_p}{r_s} \right|$$

  for cylinder type of wall heat structure.

  In PBHeatExchanger, it has

$$r_p = r_s + \delta_{wall}$$

  where δ_{wall} is wall thickness (see wall_thickness), and r_s is specified via radius_i. Thus, it requires that the primary-side is to be set at the shell-side of the heat exchanger.

- end_elems_refinement

  This same input parameter is defined in both PBOneDFluidComponent and HeatStructure. If specified, the specified value will be passed to both PBOneDFluidComponent (the primary and secondary pipes) and HeatStructure to create finer meshes at the two ends of corresponding meshes.

  This parameter is especially useful when the simulated fluid temperature experiences unphysical spatial oscillations near the inlet/outlet of PBHeatExchanger due to coarse mesh being used.

4.3.16 PBTDJ

PBTDJ is an inlet boundary component in which the flow velocity and temperature are provided by user-defined values or (time-dependent) functions. It provides boundary conditions to the connecting 1-D fluid components. Its input parameters are listed as follows:

```
[./PBTDJ]
  S_bc              = (no_default)  # Given passive scalar value on boundary
  S_fn              = (no_default)  # Name of the scalar function
  T_bc              = (no_default)  # Desired temperature
  T_fn              = (no_default)  # Name of the temperature function
  eos               = (required)    # The name of equation of state object to use.
  fluid_conduction  = (no_default)  # if modeling the fluid axial conduction
  input             = (no_default)  # Names of the connected components
  input_parameters  = (no_default)  # Name of the ComponentInputParameters user object
  v_bc              = (no_default)  # Desired velocity
  v_fn              = (no_default)  # Name of the velocity function
  wall_bc           = 0             # true for modeling a wall bc, dead end
  weak_bc           = 0             # true for weakly imposed BCs, false for
                                    # strongly imposed BCs
[../]
```

- eos (required)

  The name of equation of state object to use for this PBTDJ component.

- T_bc

  This input parameter specifies the inlet temperature, as a number, of this PBTDJ component.

- T_fn

  This input parameter specifies the inlet temperature, as a function, of this PBTDJ component. If both T_bc and T_fn are specified, T_fn is used.

- v_bc

  This input parameter specifies the inlet velocity, as a number, of this PBTDJ component.

- v_fn

  This input parameter specifies the inlet velocity, as a function, of this PBTDJ component. If both v_bc and v_fn are specified, v_fn is used.

- input

  Name of the connected component and the connected end, (in) or (out), of the component, e.g., input = 'pipe1(in)', and input = 'pipe2(out)'.

- S_bc and S_fn

  When passive scalars are present in the system, boundary conditions are also needed for them at this PBTDJ component. It first seeks a user input of S_fn, a list of function names to specify the boundary values of all scalars. If S_fn is not given, it then seeks S_bc, a list of values to specify the boundary values of all scalars. If neither is specified, zero values are used as the boundary conditions for all scalar variables.

- fluid_conduction

  This input parameter specifies if axial fluid conduction should be modeled. If not specified, it looks for the same input parameter in the global parameter inputs, see section 4.1.

- wall_bc

  This input parameter specifies if a wall boundary condition, i.e., a dead end, should be modeled using this PBTDJ component. The default value is false.

- weak_bc

  This input parameter specifies if a weakly imposed boundary condition should be used. The default value is false.

4.3.17 PBTDV

PBTDV is a boundary component in which the pressure and temperature are provided by user-defined (time-dependent) functions. It provides boundary conditions to the connecting 1-D fluid components. Note if the flow is flowing into the PBTDV, the temperature boundary condition will not be used by the connecting fluid components. Its input parameters are listed as follows:

```
[./PBTDJ]
  S_bc             = (no_default)  # Given passive scalar value on boundary
  S_fn             = (no_default)  # Name of the passive scalar function
  T_bc             = (no_default)  # Given temperature on boundary
  T_fn             = (no_default)  # Name of the temperature function
  eos              = (required)    # The name of equation of state object to use.
  fluid_conduction = (no_default)  # if modeling the fluid axial conduction
  input            = (no_default)  # Names of the connected components
  input_parameters = (no_default)  # Name of the ComponentInputParameters user object
  p_bc             = 100000        # Given pressure on boundary
  p_fn             = (no_default)  # Name of the pressure function
  stagnant         = 0             # true for modeling a stagnant back pressure
  wall_bc          = 0             # true for modeling a wall bc, dead end
  weak_bc          = 0             # true for weakly imposed BCs, false for
                                   # strongly imposed BCs
[../]
```

- eos (required)

  The name of equation of state object to use for this PBTDJ component.

- T_bc

  This input parameter specifies the inlet temperature, as a number, of this PBTDV component. It only matters when the flow direction is from PBTDV to its connected component.

- `T_fn`

  This input parameter specifies the inlet temperature, as a function, of this PBTDV component. If both `T_bc` and `T_fn` are specified, `T_fn` is used.

- `p_bc`

  This input parameter specifies the pressure, as a number, of this PBTDV component.

- `p_fn`

  This input parameter specifies the pressure, as a function, of this PBTDV component. If both `p_bc` and `p_fn` are specified, `p_fn` is used.

- `stagnant`

  This input parameter specifies if stagnant pressure boundary condition should be used in this PBTDV component.

- `input`

  Name of the connected component and the connected end, (in) or (out), of the component, e.g., `input = 'pipe1(in)'`, and `input = 'pipe2(out)'`.

- `S_bc` and `S_fn`

  The same as discussed in PBTDJ, see section 4.3.16.

- `fluid_conduction`

  This input parameter specifies if axial fluid conduction should be modeled. If not specified, it looks for the same input parameter in the global parameter inputs, see section 4.1.

- `wall_bc`

  This input parameter specifies if a wall boundary condition, i.e., a dead end, should be modeled using this PBTDV component. The default value is false.

- `weak_bc`

  This input parameter specifies if a weakly imposed boundary condition should be used. The default value is false.

4.3.18 PressureOutlet

PressureOutlet provides a subset of functionality of PBTDV, and will be removed in the future.

4.3.19 CoupledTDV

CoupledTDV is a special PBTDV component that is designed to facilitate the coupling between SAM and external CFD codes. Instead of user-specified values, in coupled code simulations, its boundary condition values are obtained from other external codes, and meanwhile, it also provides boundary conditions to these external codes. Compared with PBTDV, it does not require additional inputs, however, the code-to-code coupling is realized using CoupledCFDExecutioner.

4.3.20 CoupledPPSTDJ

CoupledPPSTDJ is a special PBTDJ component that is designed to facilitate MultiApp simulations. Instead of user-specified values, in MultiApp simulations, its boundary condition values are obtained from other MOOSE applications, and meanwhile, it also provides boundary conditions to other MOOSE applications. Inherited from PBTDJ, it requires two extra input parameters to facilitate MultiApp information passing between MOOSE applications.

- `postprocessor_vbc` and `postprocessor_Tbc`

  This input parameter specifies a Postprocessor name, which will be used to specify the velocity (temperature) boundary condition values of this CoupledPPSTDJ component.

4.3.21 CoupledPPSTDV

Similar to CoupledPPSTDJ, CoupledPPSTDV is a special PBTDV component that is designed to facilitate MultiApp simulations. Instead of user-specified values, in MultiApp simulations, its boundary condition values are obtained from other MOOSE applications, and meanwhile, it also provides boundary conditions to other MOOSE applications. Inherited from PBTDV, it requires two extra input parameters to facilitate MultiApp information passing between MOOSE applications.

- `postprocessor_pbc` and `postprocessor_Tbc`

  This input parameter specifies a Postprocessor name, which will be used to specify the pressure (temperature) boundary condition values of this CoupledPPSTDJ component.

4.3.22 PBSingleJunction

PBSingleJunction is a special junction component, and it models a zero-volume flow joint where only two 1-D fluid components are connected. It thus does not need to model the mass, momentum, and energy conservations at the junction, but to assure that the two connecting nodes (1 and 2) have consistent boundary conditions.

Its input parameters are listed as follows:

```
[./PBSingleJunction]
  eos             = (required)    # The name of equation of state object to use.
  inputs          = (no_default)  # Inputs of this junction
  nodal_Tbc       = true          # If applying temperature NodalBC to the connected pipe ends
  outputs         = (no_default)  # Outputs of this junction
  K               = 0             # The form loss coefficient
  K_reverse       = (no_default)  # The form loss coefficient in reverse direction
  weak_constraint = true          # Option to use weak formulation of single junction model
  K_Borda_Carnot  = false         # Option to use Borda-Carnot equation to calculate the form
                                  # loss coefficient due to abrupt area change
[../]
```

- `eos (required)`

  The name of equation of state object to use.

- `inputs` and `outputs`

These input parameters specify the inputs and outputs connection of this PBSingleJunction component. The input syntax is, for example, `inputs = ‘pipe1(in)’`, and `outputs = ‘pipe2(out)’`. There is only one inputs and one outputs allowed for this PBSingleJunction component.

- `weak_constraint`

  There are currently two options in the modeling of the single junction: weak formulation and strong formulation. In the weak formulation model, the single junction does not model the mass, momentum, and energy conservation at the junction, but to assure that the two connecting nodes (1 and 2) have consistent boundary conditions. This weak formulation is computationally efficient but does not enforce mass and energy balance well. The strong formulation adds the mass, momentum, and energy conservation at the junction and helps enforce the mass and energy balance. `Caution`: the weak formulation will be removed in the future.

- `nodal_Tbc`

  This input parameter specifies if temperature NodalBC to be applied to the connected pipe ends. The default value is true.

- `K` and `K_inverse`

  Those input parameters specify the form loss coefficient in the flow direction and in the reverse flow direction across this junction.

- `K_Borda_Carnot`

  The option to use the built-in Borda-Carnot equation for calculating the form loss coefficient in the forward and reverse flow direction across this junction. The form loss coefficient is calculated based on the upwind flow kinetic energy. `Caution`: if this option is set to true, the user-provided `K` and `K_inverse` will be ignored.

4.3.23 PBBranch

PBBranch models a 0-D flow junction where multiple 1-D fluid components are connected. The component assumes no volume, and thus there is no thermal inertia.

Its input parameters are listed as follows:

```
[./PBBranch]
  Area           = (required)    # Reference area of this branch
  K              = (required)    # Form loss coefficients
  K_B            = (no_default)  # coefficient B in calculating Reynolds number-dependent
                                 # form loss coefficients
  K_B_reverse    = (no_default)  # coefficient B in calculating Reynolds number-dependent
                                 # form loss coefficients in reverse direction
  K_C            = (no_default)  # coefficient C in calculating Reynolds number-dependent
                                 # form loss coefficients
  K_C_reverse    = (no_default)  # coefficient C in calculating Reynolds number-dependent
                                 # form loss coefficients in reverse direction
  K_reverse      = (no_default)  # Form loss coefficients in reverse direction
  eos            = (required)    # The name of equation of state object to use.
  initial_P      = (no_default)  # Initial pressure of this branch
  initial_T      = (no_default)  # Initial temperature of this branch
  initial_V      = (no_default)  # Initial velocity of this branch
  inputs         = (no_default)  # Inputs of this junction
  joint_model    = 1             # Using volume or joint model
  nodal_Tbc      = 0             # If applying temperature NodalBC to connected pipe ends
  outputs       = (no_default)  # Outputs of this junction
  scale_factors = '1 1 1e-06'   # variable scale factor
[../]
```

- `eos` (required)

  The name of equation of state object to use.

- `Area` (required)

  The reference area of this branch.

- `inputs` and `outputs`

  These input parameters specify the inputs and outputs connection of this PBBranch component. The input syntax is, for example, `inputs = 'pipe1(out) pipe2(out)'`, and `outputs = 'pipe3(in)'`.

- `K` (required)

  This input parameter specifies a list of values for forward form loss coefficients at each connection of this PBBranch component. The total number of listed values has to be the same as the total number of connections. The forward direction is defined as if it flows from its connected 'inputs' pipes to this PBBranch component, or from this PBBranch component to its connected 'outputs' pipes. Otherwise, the flow is in reversed direction.

- `K_reverse`

  Similar to `K`, this input parameter specifies the reverse flow form loss coefficients at each connection of this PBBranch component. It is not required, and if not specified, they assume the same values from the forward form loss coefficients. If a user input is given, the total number of listed values has to be the same as the total number of connections.

- `K_B` and `K_C`

  These two input parameters supplement input parameter, `K`, when forward form loss coefficients are Reynolds number-dependent,

  $$K_{total} = A + B\mathrm{Re}^C$$

  in which, A is the value from input parameter `K`, B from `K_B`, and C from `K_C`. If specified, these two parameters have to be both given.

- `K_B_reverse` and `K_C_reverse`

  These two input parameters are similar to `K_B` and `K_C`.

- `nodal_Tbc`

  This input parameter specifies if temperature NodalBC to be applied to the connected pipe ends. The default value is true.

- `initial_P`, `initial_V`, and `initial_T`

  These input parameters specify the initial pressure (velocity, temperature) of this PBBranch component. If not specified, the component seeks the global initial values, see section 4.1.

- scale_factors

  Similar to the global scaling factors, this input parameter specifies the local scaling factors for the three variables: pressure, velocity, and temperature. The default values are '1.0 1.0 1.0e-6'.

- joint_model (advanced)

  This input parameter specifies if volume or joint model to be used in this PBBranch component. The default value is true.

4.3.24 PBVolumeBranch

PBVolumeBranch is a type of PBBranch while considering its volume effects, and thus, it accounts for the mass and energy balance between inlets and outlets, as well as its own volume. Inherited from PBBranch, PBVolumeBranch requires additional input parameters.

```
[./PBVolumeBranch]
  Area          = (required)    # See PBBranch
  K             = (required)    # See PBBranch
  K_B           = (no_default)  # See PBBranch
  K_B_reverse   = (no_default)  # See PBBranch
  K_C           = (no_default)  # See PBBranch
  K_C_reverse   = (no_default)  # See PBBranch
  K_reverse     = (no_default)  # See PBBranch
  eos           = (required)    # See PBBranch
  initial_P     = (no_default)  # See PBBranch
  initial_T     = (no_default)  # See PBBranch
  initial_V     = (no_default)  # See PBBranch
  inputs        = (no_default)  # See PBBranch
  joint_model   = 1             # See PBBranch
  nodal_Tbc     = 0             # See PBBranch
  outputs       = (no_default)  # See PBBranch
  scale_factors = '1 1 1e-06'   # See PBBranch

  Steady        = 0             # for steady state initialization
  center        = (required)    # geometric center of the volume
  display_pps   = 0             # display post processors
  height        = (no_default)  # Height of the component
  orientation   = '0 0 1'       # Orientation vector of the component
  position      = '0 0 0'       # Origin (start) of the component
  rotation      = 0             # Rotation of the component (in degrees)
  volume        = (required)    # Volume of the component
  width         = (no_default)  # Width of the component
[../]
```

- center (required)

  The geometric center of the volume, which is important to compute pressure jump between this PBVolumeBranch and its connected pipes due to gravity head. It also overrides the values given in position (if ever specified).

- volume (required)

  The Volume of the component.

- display_pps

PBVolumeBranch adds several Postprocessors to monitor its field variables (pressure, temperature, density, and velocity). If specified true, these Postprocessors values will be displayed. The default value is false.

- `Steady`

  This input parameter specifies if the initial values are to be used for steady state initialization. The default value is false.

- `width` and `height`

  For display purpose, these parameters specify the width and height of this PBVolumeBranch component. For width, if not specified, it is computed as the pipe diameter as if the PBVolumeBranch was a round pipe section, i.e., width $= 2\sqrt{A/\pi}$, in which A is the reference area. For height, if not specified, it is computed as volume divided by the reference area.

- `orientation`, `position`, and `rotation`

  See PBOneDFluidComponent, section 4.3.1. In this component, they are used to generate mesh for display purpose.

4.3.25 Valve

Valves are mechanical devices that regulate, direct, or control the fluid flow by opening, closing, or partially blocking the flow path. There exist many types of valves, such as ball valves, check valves, pressure relief valves, etc. In nuclear reactors, examples include the main steam isolation valve in boiling water reactor designs, the pressure relief valve installed on the pressurizer of pressurized water reactor designs, and many other types of valves. The Valve component currently implemented in the code is a relatively simple one that only performs opening and closing functions. Similar to other Junction/Branch type of components, it connects pipes and regulates the flow by adjusting its opening area. The Valve component accepts users input to control its opening/closing action, and the form loss is automatically computed using the Borda-Carnot correlation [16].

Input parameters for the Valve component is listed as follows,

```
[./Valve]
  eos           = (required)    # See PBBranch
  initial_P     = (no_default)  # See PBBranch
  initial_T     = (no_default)  # See PBBranch
  initial_V     = (no_default)  # See PBBranch
  inputs        = (no_default)  # See PBBranch
  joint_model   = 1             # See PBBranch
  nodal_Tbc     = 0             # See PBBranch
  outputs       = (no_default)  # See PBBranch
  scale_factors = '1 1 1e-06'   # See PBBranch

  control_function        = (no_default) # User specified control function for the valve
  direct_control_function = (no_default) # User specified area change as a function of time
  initial_area            = (required)   # Initial flow area for the valve if it does not
                                         # start fully open
  open_area               = (required)   # Flow area when the valve is fully open
  time_constant           = (no_default) # Time constant that specifies the normalized
                                         # [0-1] rate per second
[../]
```

For these input parameters, only those additional to the PBBranch is discussed.

- open_area (required) and initial_area (required)

  The open_area input parameter is the fully opened valve area, and initial_area is the initial valve opening area.

- direct_control_function

  There are two ways to control the Valve component opening and closing. The first option is to directly control the normalized valve opening area by specifying a time-dependent direct control function, i.e., direct_control_function. The other option is to specify the opening and closing action, see control_function. The simplest direct control could be a 'PiecewiseLinear' function that linearly opens/closes the valve area. As an example, the following function demonstrates that a valve is linearly closing between 5 to 7 sec., staying as fully closed between 7 to 13 sec., and then reopens between 13 to 15 sec., and eventually stays fully opened after 15 sec.

```
[Functions]
  [./direct_control]
    # 'x' means time without explicitly defining it.
    type = PiecewiseLinear
    x = '0    5    7    13    15   1e5'
    y = '1    1    0    0     1    1'
  [../]
[]

[Components]
  [./valve_example]
    type = Valve
    ...
    direct_control_function = direct_control
    ...
  [../]
[]
```

  One can also take advantages of the smoothness of hyperbolic functions to specify a smooth opening/closing function, as demonstrated in the following example:

```
[Functions]
  [./direct_control]
    type  = ParsedFunction
    value = '1 / (1 + exp(40 * (t - 2.0)))'
  [../]
[]
```

- control_function and time_constant

  This is the second option to control the Valve, which specifies time-dependent opening/closing action of the Valve, i.e., control_function. This function must be a time-dependent 'PiecewiseConstant' function, with its values being one of 1 (performing opening action), -1 (performing closing action), and 0 (no action). This parameter is paired with an additional input parameter, time_constant, which specifies how fast the valve opening/closing action is, in [1/sec.]. In other words, the reciprocal of time_constant is the time needed to perform an opening action from fully close status to fully open status, or vice versa.

  The following example shows how the Valve is controlled using both control_function and time_constant input parameters. This control is equivalent to the 'PiecewiseLinear' example discussed in the previous direct_control_function input parameter.

```
[Functions]
  [./action_control]
    type = PiecewiseConstant
    # [0, 5], [7, 13], [15, 1e5] no action
    # [5, 7]    performing closing action
    # [13, 15]  performing opening action
    x = '0   5   7   13   15  1e5'
    y = '0   -1  0   1    0   0'
  [../]
[]

[Components]
  [./valve_example]
    type = Valve
    ...
    control_function = action_control
    time_constant    = 0.5
    ...
  [../]
[]
```

4.3.26 PBLiquidVolume

PBLiquidVolume is a special PBVolumeBranch, in which the liquid volume can change due to in and out flows, and the liquid level is tracked during the transient. The reference gas phase pressure in the PBLiquidVolume is either an ambient pressure, figure 4.12 (a), or comes from an external component, CoverGas, figure 4.12 (b). For CoverGas component, see section 4.3.27.

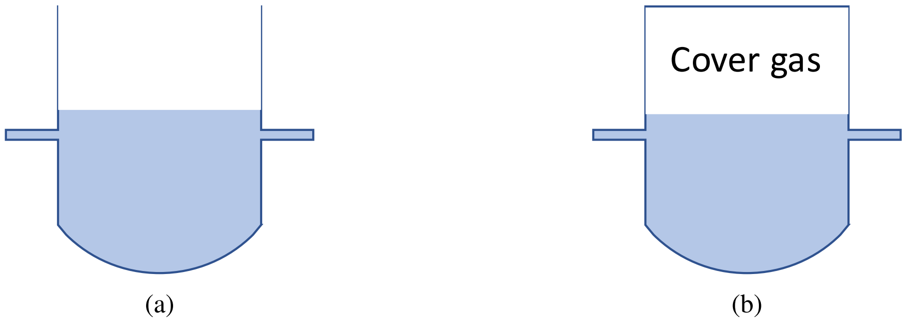


Figure 4.12: The PBLiquidVolume concept used in SAM. (a) PBLiquidVolume with ambient pressure as its reference pressure; (b) PBLiquidVolume with an external CoverGas to specify its reference pressure.

Additional to what are needed in PBVolumeBranch (see section 4.3.24), PBLiquidVolume requires several additional input parameters for the initial liquid level and the external CoverGas component, or an ambient pressure.

- `initial_level (required)`

  The initial liquid level in this PBLiquidVolume component.

- `covergas_component`

The name of the external CoverGas component, the pressure of which will be used as the reference pressure in this PBLiquidVolume component. If not specified, PBLiquidVolume takes an ambient pressure as the reference pressure. To compute the liquid phase pressure, a hydrostatic pressure head will be added to this reference pressure. The hydrostatic pressure head is calculated as $\rho g L$, with ρ the liquid density, g=9.81 the gravity constant, and L the liquid level.

- `ambient_pressure`

  If a CoverGas component is not given to provide the reference pressure, an ambient pressure is then needed as the reference pressure. The default value is 1 bar (10^5 Pa).

4.3.27 CoverGas

CoverGas component is always used together with PBLiquidVolume component, see figure 4.12 (b). It models a 0-D gas volume that is connected to one or multiple liquid volumes. The gas volume is modeled as an ideal gas, and the heat transfer between the cover gas and the liquid volumes is neglected. Its volume change is decided by the volume changes of all connecting liquid volumes.

```
[./CoverGas]
  coupled_liquid_volume = 0            # If the connected liquid volume is coupled with
                                       # an external app.
  g                     = 9.81         # gravity acceleration constant
  gamma                 = 1.66667      # gamma value (cp/cv)
  initial_P             = (required)   # Initial pressure in the gas
  initial_T             = (required)   # Initial temperature in the gas
  initial_Vol           = (required)   # Initial volume of the cover gas
  n_liquidvolume        = (required)   # Number of connecting liquid volumes
  name_of_liquidvolume  = (required)   # liquid volumes names
[../]
```

- `initial_P (required)` and `initial_T (required)`

  The initial pressure and temperature of the gas phase.

- `initial_Vol (required)`

  The initial volume of the gas phase.

- `n_liquidvolume (required)`

  Number of connected liquid volumes.

- `name_of_liquidvolume (required)`

  A list of names of connected PBLiquidVolume components. The total number of these connected PBLiquidVolume components has to be the same as `n_liquidvolume`. The changes of the liquid volume in these connected PBLiquidVolume will be used to determine the gas phase volume in this CoverGas component, consequently, its pressure.

- `gamma`

  Gamma value to be used in the ideal gas equation of state that computes the gas phase pressure as its volume changes. The default value is 1.66667.

- `coupled_liquid_volume`

  This input parameter specifies if the connected liquid volume is coupled with an external app. The default value is false.

- `g`

  Gravity acceleration constant. The pressure difference between the gas phase and its connected PBLiquidVolume liquid phase is $\rho g L$, with ρ the liquid density, g the gravity acceleration constant, and L the liquid level in PBLiquidVolume component.

4.3.28 PBPump

PBPump is another special junction component, and it simulates a pump, in which the pump head can be dependent on a pre-defined function, or can be automatically adjusted to match user-specified mass flow rate. It inherits from the PBBranch component, and therefore assumes the volume of the pump is neglectable. More complex pump models will be developed in future SAM enhancements. Pumping power can be modeled and considered in the energy conservation of the junction.

Its input parameters are listed as follows:

```
[./PBPump]
  Area          = (required)    # See PBBranch
  K             = (required)    # See PBBranch
  K_B           = (no_default)  # See PBBranch
  K_B_reverse   = (no_default)  # See PBBranch
  K_C           = (no_default)  # See PBBranch
  K_C_reverse   = (no_default)  # See PBBranch
  K_reverse     = (no_default)  # See PBBranch
  eos           = (required)    # See PBBranch
  initial_P     = (no_default)  # See PBBranch
  initial_T     = (no_default)  # See PBBranch
  initial_V     = (no_default)  # See PBBranch
  inputs        = (no_default)  # See PBBranch
  joint_model   = 1             # See PBBranch
  nodal_Tbc     = 0             # See PBBranch
  outputs       = (no_default)  # See PBBranch
  scale_factors = '1 1 1e-06'   # See PBBranch


  Head          = 0             # Pump head
  Head_fn       = (no_default)  # Name of the pressure head function
  pump_heating  = 0             # if pump heating is included in the energy conservation

  Desired_mass_flow_rate    = (no_default)  # The desired mass flow rate of the pump.
                                            # flow rate and the desired one.
  Mass_flow_rate_tolerance = 0.0001        # Relative tolerance between the pump delivered
                                            # mass flow rate and the desired one.
  Response_interval         = 1             # The time interval between two consecutive pump
                                            # head adjustments.
[../]
```

- `Head`

  This input parameter specifies the pump head value, in [Pa]. The default value is 0.

- `Head_fn`

  This input parameter specifies the function name to compute pump head value.

- pump_heating

  This input parameter specifies if pump heating effect should be considered in energy balance. The default is false.

- Desired_mass_flow_rate

  This input parameter accepts a user-specified mass flow rate, such that the pump will automatically adjust the pump head to match this value. If specified, the pump head specified in the Head input parameter is used as the initial guessing value to start with the automatical adjustment.

- Mass_flow_rate_tolerance

  When a user-specified mass flow rate is given, this input parameter the absolute relative tolerance between the real pump mass flow rate compared with the user-specified one. If within this tolerance, the pump stops to adjust its pump head as it is deemed that the desired mass flow rate already achieved. The default value is 10^{-4}.

- Response_interval

  When a user-specified mass flow rate is given, this input parameter specifies the how fast, i.e., the time internal between two consecutive pump head adjustments. The default value is 1 second.

4.3.29 StagnantVolume

StagnantVolume models a stagnant liquid volume, which has no connections to 1-D fluid components but is allowed to connect to a 0-D volume or 1-D or 2-D heat structures for heat transfer. It is assumed that there is no net mass transfer between StagnantVolume and the connecting 0-D volumes. Its input parameters are listed as follows:

```
[./StagnantVolume]
  center          = (required)   # geometric center of the volume
  coupled_volume  = (no_default) # Coupled volume component name
  eos             = (required)   # The name of equation of state object to use.
  height          = (no_default) # Height of the component
  initial_T       = (required)   # initial temperature of the component
  mass_flow       = 0.           # Function name for the exchanged flow between volumes
  orientation     = '0 0 1'      # Orientation vector of the component
  position        = '0 0 0'      # Origin (start) of the component
  rotation        = 0            # Rotation of the component (in degrees)
  volume          = (required)   # Volume of the component
  width           = (no_default) # Width of the component
[../]
```

- center (required)

  The geometric center of the volume. It overrides the values given in position (if ever specified), and is used to generate mesh for display purpose.

- eos (required)

  The name of equation of state object to use.

- initial_T (required)

  The initial fluid temperature of the component.

- `height` and `width`

  These input parameters specify the height and width of the component, both of which are to be used to generate mesh for display purpose. The default value for both parameters are 1 m.

- `coupled_volume`

  The name of coupled volume, e.g., another StagnantVolume, which this StagnantVolume component exchanges energy with. The energy exchange between these two volumes is computed as $\dot{m}\bar{c_p}\Delta T$, in which $\dot{m}$ is the mixing mass flow rate (see `mass_flow`), $\bar{c_p}$ is the specific heat computed at the average temperature between the two mixing volume, ΔT the temperature difference between the two mixing volume.

- `mass_flow`

  If a coupled volume is specified, this input parameter specifies the mixing mass flow rate between the two mixing volume. It can be either a number or a function name.

- `orientation`, `position`, and `rotation`

  Parameters not used.

4.3.30 LiquidTank

The LiquidTank component of SAM simulates a PBVolumeBranch (or PBLiquidVolume) and the heat structure (modeled as PBCoupledHeatStructure) attached to it in order to capture this additional thermal inertia. The input parameters of the LiquidTank component requires those to describe the PBVolumeBranch (or PBLiquidVolume) and those to describe the PBCoupledHeatStructure attached to it.

The LiquidTank component automatically create a PBLiquidVolume component, if a CoverGas component is connected to determine its gas phase pressure; otherwise, a PBVolumeBranch component is create. It assumes that the PBVolumeBranch (or PBLiquidVolume) is connected to the left-side surface of PBCoupledHeatStructure, and additional boundary condition input parameters are required for the right-side surface of PBCoupledHeatStructure.

The list of input parameters are given in the following list. Part of them are required to describe the PBVolumeBranch (or PBLiquidVolume) component, which could be referred to section 4.3.24 (or section 4.3.26); and part of them are required to describe the PBCoupledHeatStructure component, which could be referred to section 4.3.12.

```
[./LiquidTank]
  Area                          = (required)         # See PBVolumeBranch (PBBranch)
  K                             = (required)         # See PBVolumeBranch (PBBranch)
  Steady                        = 0                  # See PBVolumeBranch
  center                        = (required)         # See PBVolumeBranch
  display_pps                   = 0                  # See PBVolumeBranch
  eos                           = (required)         # See PBVolumeBranch
  height                        = (no_default)       # See PBVolumeBranch
  initial_P                     = (no_default)       # See PBVolumeBranch (PBBranch)
  initial_T                     = (no_default)       # See PBVolumeBranch (PBBranch)
  initial_V                     = (no_default)       # See PBVolumeBranch (PBBranch)
  inputs                        = (no_default)       # See PBVolumeBranch
  rotation                      = 0                  # See PBVolumeBranch
  scale_factors                 = '1 1 1e-06'        # See PBVolumeBranch
  volume                        = (required)         # See PBVolumeBranch
  width                           = (no_default)       # See PBVolumeBranch
  nodal_Tbc                       = 1                  # See PBVolumeBranch (PBBranch)
  orientation                     = '0 0 1'            # See PBVolumeBranch
  outputs                         = (no_default)       # See PBVolumeBranch
  position                        = '0 0 0'            # See PBVolumeBranch

  HS_BC_type_right                = (required)    # See HS_BC_type of PBCoupledHeatStructure
  HT_surface_area_density_right = (no_default)         # See PBCoupledHeatStructure
  Hw                              = (no_default)  # See Hw_left of PBCoupledHeatStructure
  Hw_right                        = (no_default)       # See PBCoupledHeatStructure
  T_amb_right                     = 300                # See PBCoupledHeatStructure
  T_bc_right                      = 600                # See PBCoupledHeatStructure
  Ts_init                         = (no_default)       # See PBCoupledHeatStructure
  dim_hs                          = 2                  # See PBCoupledHeatStructure
  elem_number_axial               = 1                  # See PBCoupledHeatStructure
  elem_number_radial              = (required)         # See PBCoupledHeatStructure
  heat_source_solid               = 0                  # See PBCoupledHeatStructure
  heat_transfer_area              = (no_default)       # See heat_transfer_area_left
                                                       # of PBCoupledHeatStructure
  heat_transfer_area_right        = (no_default)       # See PBCoupledHeatStructure
  hs_names                        = (no_default)       # See PBCoupledHeatStructure
  hs_type                         = cylinder           # See PBCoupledHeatStructure
  length                          = (required)         # See PBCoupledHeatStructure
  material_hs                     = (required)         # See PBCoupledHeatStructure
  name_comp_right                 = (no_default)       # See PBCoupledHeatStructure
  qs_right                        = (no_default)       # See PBCoupledHeatStructure
  radius_i                        = (no_default)       # See PBCoupledHeatStructure
  width_of_hs                     = (required)         # See PBCoupledHeatStructure

  initial_level                   = (no_default)       # See PBLiquidVolume
  covergas_component              = (no_default)       # See PBLiquidVolume

  mesh_disp_gap                   = 0.1                # Axial offset for mesh generation
[../]
```

Some details of the input parameters as discussed as follows.

- mesh_disp_gap

  This input parameter specifies mesh offset in the y-direction, with respect to the fluid component mesh, when creating heat structure meshes. The default value for this parameter is 0.1 [m]. If ‘cylinder‘ is specified for hs_type, this mesh offset value will be overridden by half of radius_i value.

4.3.31 ReactorCore

The ReactorCore component describes a pseudo three-dimensional reactor core by connecting bypass channels to their neighboring core channels (with duct walls). Its input parameters are listed below.

```
[./ReactorCore]
  n_bypasschan       = (required)   # Number of BypassChannels
  n_corechan         = (required)   # Number of CoreChannels
  name_of_bypasschan = (required)   # BypassChannel names
  name_of_corechan   = (required)   # CoreChannel names
[../]
```

- n_bypasschan (required)

  The total number of bypass channels in this ReactorCore component.

- n_corechan (required)

  The total number of core channels in this ReactorCore component.

- name_of_bypasschan (required)

  The names of all bypass channels.

- name_of_corechan (required)

  The names of all core channels.

4.3.32 SurfaceCoupling

The SurfaceCoupling component models the heat transfer between two solid surfaces, suitable for radiation heat transfer or gap heat transfer between them.

```
[./SurfaceCoupling]
  area_ratio         = 1                        # Area ratio between the two surfaces
  coupling_type      = RadiationHeatTransfer(required) # Heat transfer coupling type
  eos                = (no_default)                     # The name of EOS to use
  epsilon_1          = 1                        # Surface 1 emissivity
  epsilon_2          = 1                        # Surface 2 emissivity
  h_gap              = (no_default)             # gap conductance
  length             = (no_default)             # gap length
  radius_1           = (no_default)             # Surface 1 radius
  surface1_name      = (required)               # The name of the Surface 1
  surface2_name      = (required)               # The name of the Surface 2
  use_displaced_mesh = 1                        # Whether or not this object should use the
                                                # displaced mesh for computation.
  view_factor        = 1                        # View factor from surface master (1) to
                                                # surface slave (2)
  width              = (no_default)             # gap width
[../]
```

- surface1_name (required) and surface2_name (required)

  The name of surface 1 (2) that participates in the radiation or gap heat transfer.

- coupling_type (required)

  The heat transfer mechanism type of the heat transfer, either ‘RadiationHeatTransfer’ for radiation heat transfer, or ‘GapHeatTransfer’ for gap heat transfer. The default value is ‘RadiationHeatTransfer’.

- area_ratio

  The ratio of surface 1 area to surface 2 area, which is only required for radiation heat transfer to compute the heat flux between the two surfaces. If not specified, the default value is 1.

- radius_1

  The radius of surface 1, if surface 1 is of cylindrical type.

- epsilon_1 and epsilon_2

  The emissivity of surface 1 (2), only required for radiation heat transfer mechanism. Both parameters have the same default value 1.

- view_factor

  This parameter defines the view factor between surfaces 1 and 2, only required for radiation heat transfer mechanism. The default value is 1.

- h_gap

  For gap heat transfer mechanism, if user-specified value is desired for the gap heat transfer coefficient, this input parameter specifies such a value. If not specified, SAM will compute the gap heat transfer coefficient from other input parameters.

- eos

  The equation of state that will be used to compute gap heat transfer coefficient, only required when the heat transfer mechanism is gap heat transfer, and when user-specified heat transfer coefficient is not given.

- width and length

  Gap width (length), only required when the heat transfer mechanism is gap heat transfer, and when user-specified heat transfer coefficient is not given.

- use_displaced_mesh

  This parameter specifies that if displaced mesh to be used. The default value is true, and it is safe to use this default value.

4.3.33 ReactorPower

ReactorPower is a non-geometric component for describing the total reactor power, which can be dependent on either user-defined functions (such as describing the decay heat curve), or computed externally from SAM's PointKinetics component (see section 4.3.34). The total reactor power variable is used in core components such as PBCoreChannel and PBBypassChannel.

```
[./ReactorPower]
  initial_power            = (required)     # Initial total power
  initial_fission_power    = (no_default)   # Initial prompt fission power
  power_history            = (no_default)   # function name for power
  enable_decay_heat        = (no_default)   # Indicate whether to employ decay
                                            # heat model for stationary fuel
  isotope_fission_fraction = (no_default)   # A vector of length four, containing fractions
                                            # of all fissions from U-235, U-238,
                                            # Pu-239, and Pu-241.
  operating_power          = (no_default)   # operating power used to initialize decay heat
  decay_heat               = (no_default)   # Function (name) that provides decay heat curve
  decay_heat_channel_name  = (no_default)   # Define the channel/bypass names with decay
                                            # heat curves
  pke                      = (no_default)   # The name of the point kinetics component that
                                            # computes reactor power
[../]
```

- initial_power (required)

  This is the total power at the initial time. When initial_fission_power is absent, and decay heat calculation is requested. This parameter is also used to calculate the initial fission power.

- `initial_fission_power` (required)

  This is the (prompt) fission power at the initial time. This is used to initialize the fission power calculation using PKE. If the decay heat calculation is requested, this term, if present, will be used to calculate the initial power, i.e., to replace the value read from the initial power.

- `power_history` (required)

  This takes the name of a time-dependent power function. The power function will be multiplied by the initial power to calculate the power at a given time, and the decay heat model and the point kinetic model would not be used.

- `enable_decay_heat` (required)

  This boolean parameter is used to indicate whether to calculate the decay heat for stationary fuel. If this parameter is absent, the default is false.

- `isotope_fission_fraction` (required)

  This parameter is required when enable_decay_heat is true. The format of the input is a vector of size four. It indicates what fractions of all fissions in the reactor are coming from U-235, U-238, Pu-239, and Pu-241.

- `operating_power`

  This is the power history before $t = 0$ of the problem. It has to be a constant in the current implementation. It is used to initialize the decay heat at $t = 0$. When this parameter is not present while decay heat calculation in demanded, the initial_power will be used for initializing the decay heat.

- `decay_heat`

  If a decay heat curve is to be used to compute the reactor power, this input parameter specifies the decay heat curve function name.

- `decay_heat_channel_name`

  This input parameter specifies the core channel and/or bypass channel names with decay heat curves.

- `pke`

  This parameter takes the name of a PointKinetics component. If this parameter is present, the point kinetic equation will be used to solve the fission power as a function of time.

4.3.34 PointKinetics

The PointKinetics component is the build-in point kinetics model of SAM, which models the transient behaviors of reactor fission power, delayed-neutron precursors, as well as reactivity feedback from other components, e.g., core channels. In case of modeling molten-salt reactors, where drifting delayed neutron precursors effect cannot be ignored, PointKinetics component also take account into the net flow in (out) effect as an additional source (sink) term. The net flow in (out) effect is captured in a coupled PBMoltenSaltChannel component.

```
[./PointKinetics]
  Initial_DNP_value                 = (no_default)  # Define the initial value for delay
                                                    # neutron precursor
  LAMBDA                            = (required)    # Prompt neutron lifetime
  Moving_DNP_bypass_channels        = (no_default)  # Define the bypass channels for moving
                                                    # delay neutron precursor
  Moving_DNP_name                   = (no_default)  # Define the moving delay neutron
                                                    # precursor names
  betai                             = (required)    # Delay neutron fraction for group
                                                          # i
  core_radial_expansion_reactivity_coefficients = (no_default) # Core radial expansion
                                                               # reactivity coefficients
                                                               # (delta_k / k per kg)
  core_radial_expansion_reactivity_feedback  = 0       # Enable core radial expansion
                                                       # reactivity feedback.
  core_radial_expansion_weights           = (no_default)  # Weights for core constraint
                                                          # system on the radial expansion
                                                          # reactivity.
  core_radial_thermal_expansion_coefficient  = (no_default)  # Thermal expansion
                                                             # coefficients for
                                                             # core constraint system
                                                             # at different locations.
  coupled_radial_displacements_pps  = (no_default)  # coupled radial displacements for
                                                    # radial expansion reactivity.
  coupled_radial_temperatures_pps   = (no_default)  # coupled temperature for radial
                                                    # expansion reactivity.
  feedback_components               = (no_default)  # Components which have Thermal
                                                    # -Hydraulics feedback on reactivity
  feedback_start_time               = 0             # The time that the reactivity
                                                    # feedback starts.
  lambda                            = (required)    # Delay neutron precursor
                                                    # decay constant
  n_radial_constraint_system        = (no_default)  # Number of radial constraint system
                                                    # for core radial expansion reactivity
                                                    # feedback.
  rho_fn_name                       = (no_default)  # External reactivity (delta k per
                                                    # k)
  use_external_radial_displacement  = 0             # Enable coupled radial displacement
                                                    # from external thermo-mechanical
                                                    # module.
[../]
```

- `Delay_neutron_precursor_name` (required)

  This input parameter specifies a list of names for delayed neutron precursors.

- `Initial_DNP_value`

  This input parameter specifies a list of initial values for delayed neutron precursor populations. The total number of list values have to be the same as the total number of delayed neutron precursor names. However, if not specified, delayed neutron precursor populations are initialized as:

  $$C_{i,initial} = \frac{\beta_i}{\Lambda\lambda_i}$$

  in which, β_i is the delayed neutron precursor fraction for group i (see `betai`), Λ the prompt neutron lifetime (see `LAMBDA`), λ_i the delayed neutron precursor decay constant for group i (see `lambda`).

- `LAMBDA` (required)

  This input parameter specifies the prompt neutron lifetime.

- `lambda` (required)

  This input parameter specifies a list of decay constants for delayed neutron precursors. The total number of list values have to be the same as the total number of delayed neutron precursor names.

- `betai` (required)

  This input parameter specifies a list of delayed neutron precursor fractions. The total number of list values have to be the same as the total number of delayed neutron precursor names.

- `Normalized_fission_power` (required)

  This input parameter specifies the name of the normalized fission power variable.

- `rho_fn_name`

  This input parameter specifies the function name to introduce an external reactivity, additional to those from reactor feedbacks, to the PointKinetics model. If not specified, this external reactivity is 0.

- `feedback_components`

  Besides external reactivity function, the other main reactivity feedback mechanism is thermal-hydraulics feedback from reactor core channel components. This input parameter specifies a list of components from which reactivity feedback will be computed.

- `feedback_start_time`

  The time that reactor core channel components start to compute reactivity feedback. The default value is 0 second.

- `core_radial_expansion_reactivity_feedback`

  This input parameter specifies if core radial expansion reactivity feedback should be modeled. The default value is false. If modeled, the total reactivity feedback due to core radial expansion is computed as,

  $$\Delta R_{radialexpansion} = \sum_{i=1}^{N} \left(\frac{\Delta L}{L}\right)_i w_i \rho_i$$

  in which,

  N is the the total number of radial sections to compute reactivity feedback;
  see `n_radial_constraint_system`.

  $(\Delta L/L)_i$ is the radial core displacement value in the i-th radial section;

  w_i is the weight value of the i-th radial section; see `core_radial_expansion_weights`.

  ρ_i is core radial expansion coefficient at the i-th radial section;
  see `core_radial_expansion_reactivity_coefficients`.

Currently, SAM provides two ways to model core radial expansion, i.e., a simple built-in function or coupled from external simulations. The simpler one is a built-in function, which computes the i-th radial expansion as,

$$\left(\frac{\Delta L}{L}\right)_i = \alpha_i \left(T_i - T_{0,i}\right)$$

in which,

α_i is the thermal expansion coefficient of the constraint structure at the i-th radial section, see `thermal_expansion_coefficient`.

T_i is the core temperatures in the i-th radial section;
see `coupled_radial_temperatures_pps`.

$T_{0,i}$ is the initial, i.e., when reactor core channel components start to compute reactivity feedback, core temperatures in the i-th radial section.

The radial core displacement can also be modeled from an external code, and then coupled with SAM to compute the reactivity feedback value (`use_external_radial_displacement = true`).

- `use_external_radial_displacement`

  This input parameter specifies if core radial displacement should be modeled from an external simulation and their values are provided as coupled values. The default value is false. If specified true, externally computed core radial expansion displacement is expected from user input, see `coupled_radial_displacements_pps`.

- `n_radial_constraint_system`

  If core radial expansion reactivity feedback is modeled, this input parameter specifies the number of radial constraint system, i.e., number of core radial sections, for core radial expansion reactivity feedback.

- `core_radial_expansion_reactivity_coefficients`

  This input parameter specifies a list of core radial expansion reactivity feedback coefficient values. The total number of this list of values has to be the same as the number of core radial sections (see `n_radial_constraint_system`).

- `core_radial_expansion_weights`

  This input parameter specifies a list of values for core radial expansion reactivity feedback weights. The total number of this list of values has to be the same as the number of core radial sections (see `n_radial_constraint_system`).

- `thermal_expansion_coefficient`

  If radial expansion is modeled used SAM's built-in function, this input parameter specifies a list of thermal expansion coefficients to compute radial core displacement values. The total number of this list of values has to be the same as the number of core radial sections (see `n_radial_constraint_system`).

- `coupled_radial_temperatures_pps`

If radial expansion is modeled used SAM's built-in function, this input parameter specifies a list of Postprocessor names that compute core temperatures in each radial section. The total number of this list of names has to be the same as the number of core radial sections (see `n_radial_constraint_system`).

- `coupled_radial_displacements_pps`

  If core radial displacement should be modeled from an external simulation and their values are provided as coupled values, this input parameter specifies a list of Postprocessor names that compute core radial displacement at each radial section. The total number of this list of names has to be the same as the number of core radial sections (see `n_radial_constraint_system`).

- `Moving_DNP_bypass_channels`

  If drifting delayed neutron precursors effect should be considered, this input parameter specifies the PBMoltenSaltChannel component name, from which the net flow in (out) of drifting delayed neutron precursors are computed.

- `Moving_DNP_name`

  If drifting delayed neutron precursors effect should be considered, this input parameter specifies the names of these delayed neutron precursors.

4.3.35 ReferenceBoundary

ReferenceBoundary component provides a fixed value boundary condition to a one-dimensional fluid type of component. This boundary condition can be applied to normal flow parameters, such as pressure, velocity, and temperature, as well as scalar variables.

```
[./ReferenceBoundary]
  coupled_var      = (no_default)      # coupled variable at bc
  input            = (no_default)      # Names of the connected components
  value            = (no_default)      # Given variable value on boundary
  variable         = (required)        # variable to be set at bc
[../]
```

- `input`

  This input parameter specifies where this boundary condition should be applied, e.g., `input = 'pipe-1(in)'`.

- `variable (required)`

  This input parameter specifies which variable this boundary condition should be applied, e.g., `variable = pressure`. In principle, this can be any field variable, but pressure is commonly used to setup the system reference pressure.

- `value`

  This input parameter specifies the value to be applied to the variable in this boundary condition.

- `coupled_var`

  This input parameter specifies a coupled variable, whose value is to be applied to the variable in this boundary condition.

4.3.36 PipeChain

PipeChain is a non-geometric component for sequentially connecting a number of fluid components. It automatically generates the needed PBSingleJunction components between the specified fluid components. The purpose of this component for user friendliness.

There are only two input parameters required for this component:

- `eos` (required)

  Equation of state to be used for all automatically-generated PBSingleJunctions.

- `component_names`

  This input parameter specifies a list of N sequentially connected fluid components, and $N-1$ PBSingleJunctions will be automatically generated to connect them.

4.3.37 ChannelCoupling

ChannelCoupling is a non-geometric component for coupling two 1-D fluid components (with energy exchange). It is intended to model the flow mixing between two parallel channels.

```
[./ChannelCoupling]
  beta               = (required)      # turbulent mixing parameter
  eos                = (required)      # The name of EOS to use
  gap_width          = (required)      # The gap width
  pipe1_name         = (no_default)    # The name of the Pipe 1
  pipe2_name         = (no_default)    # The name of the Pipe 2
  var_scaling_factor = 0.001           # turbulent mixing flux variable scaling factor
[../]
```

- `pipe1_name` and `pipe2_name`

  The names of the two pipes where this flow mixing is happening.

- `eos` (required)

  Equation of state to be used in this ChannelCoupling component.

- `gap_width` (required)

  Gap width between the two pipes where this flow mixing is happening.

- `beta` (required)

  This input parameter specifies the turbulent mixing coefficient value to be used to compute the inter-channel mass flux due to turbulent mixing, which will then be used to compute the inter-channel energy flux due to turbulent mixing.

- `var_scaling_factor`

  This input parameter specifies the scaling factor for the variable for computing the inter-channel mass flux due to turbulent mixing.

4.3.38 HeatPipe

The *HeatPipe* component is used to model a conventional 3-zone cylindrical heat pipe, see Figure 4.13. Along the axial direction, the heat pipe is divided to 3 zones: evaporator, adiabatic, and condenser. Along the radial direction, the heat pipe consists of vapor core, wick, and wall blocks. In the current SAM modeling of the heat pipe, phase transition of the fluid is not considered. The vapor core is modeled as a perfect conductor with high thermal conductivity. This approximation is based on the fact that the thermal resistance in the vapor core is small under normal operation conditions.

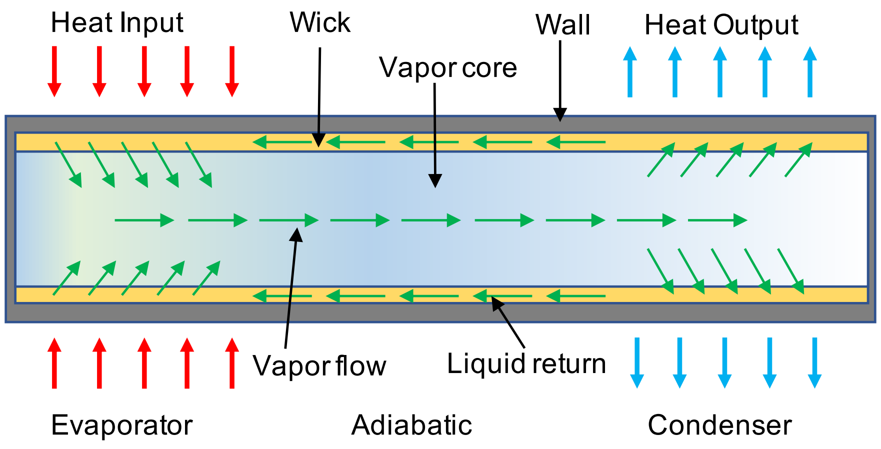


Figure 4.13: Sketch of a conventional 3-zone cylindrical heat pipe

HeatPipe is directly inherited from the base HeatStructure component. The side surface of the heat structure is modified to add the evaporator, adiabatic, and condenser wall outer surface. Boundary conditions are controlled at the evaporator and condenser wall outer surfaces, such as adiabatic, fixed temperature, convective heat transfer with ambient, coupled with 0-D liquid volume or 1-D liquid component, and gap heat transfer coupled with a solid surface.

The majority of boundary condition options of HeatPipe emulate that of PBCoupledHeatStructure, see section 4.3.12. Most input parameters of HeatPipe are inherited from the base HeatStructure component, see section 4.3.2, and additional input parameters are required to setup its evaporator(condenser) boundary conditions. The complete input parameters of HeatPipe are listed below:

```
[./<HeatPipe>]
  offset                        = '0 0 0'      # see HeatStructure
  orientation                   = '0 0 1'      # see HeatStructure
  position                      = '0 0 0'      # see HeatStructure
  power_fraction                = (no_default) # see HeatStructure
  Ts_init                       = (no_default) # see HeatStructure
  hs_names                      = (no_default) # see HeatStructure
  heat_source_solid             = 0            # see HeatStructure
  hs_power                      = (no_default) # see HeatStructure
  hs_power_shape_fn             = (no_default) # see HeatStructure

  length                        = (required)   # Total axial length
  elem_number_radial            = (required)   # Radial elements of each block
  elem_numbers_axial            = (required)   # Axial elements of each zone
```

```
material_hs                  = (required)   # Material name of each block
width_of_hs                  = (required)   # Width of each radial block
zone_lengths                 = (required)   # Length of each axial zone
elem_number_axial            = 1            # Will be ignored by HeatPipe
hs_type                      = (no_default) # Geometry type: MUST be cylinder
dim_hs                       = 2            # Dimension: MUST be 2D
radius_i                     = 0            # Default value of 0 should be used


depth_plate                  = (no_default) # Not used for HeatPipe.
axial_offset                 = 0            # Not used for HeatPipe.
end_elems_refinement         = 1            # Not used for HeatPipe.
HT_area_multiplier_left      = 1            # Not used for HeatPipe
HT_area_multiplier_right     = 1            # Not used for HeatPipe


HP_BC_type                   = 'NONE NONE'  # Boundary condition types for evaporator
                                            # and condenser wall.
HT_surface_area_density_cond = 1            # Heat transfer surface area density at
                                            # condenser surface
HT_surface_area_density_evap = 1            # Heat transfer surface area density at
                                            # evaporator surface

D_heated_cond                = (no_default) # Characteristic heated length at
                                            # condenser surface
D_heated_evap                = (no_default) # Characteristic heated length at
                                            # evaporator surface


HTC_geometry_type_cond       = Pipe         # Heat transfer geometry type at
                                            # condenser surface
HTC_geometry_type_evap       = Pipe         # Heat transfer geometry type at
                                            # evaporator surface

Hw_cond                    = (no_default) # Convective HTC at condenser surface
Hw_evap                    = (no_default) # Convective HTC at evaporator surface
T_amb_cond                 = 300          # Condenser ambient temperature
T_amb_evap                 = 300          # Evaporator ambient temperature
T_bc_cond                  = 600          # Fixed Temperature BC at condenser surface
T_bc_evap                  = 600          # Fixed Temperature BC at evaporator surface
T_external_cond            = (no_default) # Coupled variable for external temperature at
                                          # condenser surface
T_external_evap            = (no_default) # Coupled variable for external temperature at
                                          # evaporator surface
h_external_cond            = (no_default) # Coupled variable for external HTC at
                                          # condenser surface
h_external_evap            = (no_default) # Coupled variable for external HTC at
                                          # evaporator surface
eos_cond                   = (no_default) # The name of EOS to use for condenser surface
eos_evap                   = (no_default) # The name of EOS to use for evaporator surface
heat_transfer_area_cond    = 1            # Convective heat transfer area at
                                          # condenser surface
heat_transfer_area_evap    = 1            # Convective heat transfer area at
                                          # evaporator surface
name_comp_cond             = (no_default) # Liquid volume connected to condenser surface
name_comp_evap             = (no_default) # Liquid volume connected to evaporator surface
qs_cond                      = (no_default) # Heat flux at the condenser surface
qs_evap                      = (no_default) # Heat flux at the evaporator surface
qs_external_cond             = (no_default) # Coupled variable for external heat flux at
                                            # condenser surface
qs_external_evap             = (no_default) # Coupled variable for external heat flux at
                                            # evaporator surface

gap_surface_name_cond        = (no_default) # Copuled master surface name for gap heat
```

```
                                         # transfer in the condenser surface
  gap_surface_name_evap        = (no_default) # Copuled master surface name for gap heat
                                         # transferin the evaproator surface
  h_gap_cond                   = (no_default) # Gap conductance for condenser surface
  h_gap_evap                   = (no_default) # Gap conductance for evaporator surface
  length_cond                  = (no_default) # Gap length for condenser surface
  length_evap                  = (no_default) # Gap length for evaporator surface
  width_cond                   = (no_default) # Gap width for condenser surface
  width_evap                   = (no_default) # Gap width for evaporator surface
[../]
```

The explanation of most input parameters inherited from HeatStructure is discussed in section 4.3.2. The remaining input parameters are explained below:

- `length` (required)

  Total axial length of the heat pipe.

- `elem_number_radial` (required)

  The vector of radial elements for the vapor, wick, and wall blocks of the heat pipe.

- `elem_numbers_axial` (required)

  The vector of axial elements for the 3 zones (evaporator, adiabatic, and condenser) of heat pipe. The summation of elements in each zone is the total number of elements in the axial direction. The input parameter "elem_number_axial" will be ignored. Warning: notice the minor difference between "elem_numbers_axial" and "elem_number_axial".

- `material_hs` (required)

  The name of materials for the vapor core, wick, and wall blocks.

- `width_of_hs` (required)

  The width of vapor core, wick, and wall blocks.

- `zone_lengths` (required)

  The length of evaporator, adiabatic, and condenser zones. Notice that non-uniform mesh sizes are possible for different axial zones.

- `hs_type`

  The geometry type of a heat pipe must be "cylinder".

- `HS_BC_type` (required)

  This input parameter specifies the two boundary condition types for the evaporator and condenser surface of HeatPipe component. Input options include "Temperature", "Convective", "Coupled", and "GapHeatTransfer".

  If "Temperature" is specified, a Dirichlet type of boundary conditions is used, and it expects an additional user-input for the boundary condition temperature, see `T_bc_evap(cond)`.

  If "Convective" is specified, a wall heat flux will be used. The wall heat flux could be directly specified, see `qs_evap(cond)`. It is also possible to compute the wall surface flux by providing ambient temperature (see `T_amb_evap(cond)`) and heat transfer coefficient to the ambient (see `Hw_evap(cond)`).

In code coupling computation situations, wall heat flux boundary conditions could be calculated from an external code. Corresponding to the two conditions discussed above, one could specify an externally calculated wall heat flux, see `qs_external_evap(cond)`; or a combination of external temperature and wall heat transfer coefficient, see `T_external_evap(cond)` and `h_external_evap(cond)`.

If "Coupled" is specified, still a wall heat flux will be used, but the fluid temperature comes from a neighboring component, either a PBVolumeBranch or a PBOneDFluidComponent, see `name_comp_evap(cond)`. Heat transfer coefficient to the coupled component can be specified in `Hw_evap(cond)`, and if not specified, they will be automatically computed.

If "GapHeatTransfer" is specified, the evaporator/condenser surface will be coupled with other solid surface through gap heat transfer. The name of the coupled surface is specified with the input parameter `gap_surface_name_evap(cond)`. The parameters related to the gap conductance are specified with `h_gap_evap(cond)`, `length_evap(cond)`, and `width_evap(cond)`.

- `T_bc_evap` and `T_bc_cond`

  If the evaporator/condenser boundary condition type is "Temperature", it accepts the value (or a function) for the evaporator/condenser boundary condition temperatures. The default values are 600 K.

- `qs_evap` and `qs_cond`

  If the evaporator/condenser boundary condition type is "Convective", it accepts the value (or a function) for the evaporator/condenser wall heat flux as boundary conditions.

- `T_amb_evap` and `T_amb_cond`; `Hw_evap` and `Hw_cond`

  If the evaporator/condenser boundary condition type is "Convective", `T_amb_evap(cond)`) specifies the ambient temperature, and `Hw_evap(cond)`) specifies the wall heat transfer coefficient to compute the wall heat flux. Both of these inputs could be either values or function names. The default value for `T_amb_evap(cond)` is 300 K.

- `qs_external_evap` and `qs_external_cond`

  If the evaporator/condenser boundary condition type is "Convective", it accepts the variable name for externally computed evaporator/condenser wall heat flux as boundary conditions.

- `T_external_evap` and `T_external_cond`; `h_external_evap` and `h_external_cond`

  If the evaporator/condenser boundary condition type is "Convective", `T_external_evap(cond)` specifies an externally computed ambient temperature, and `h_external_evap(cond)` specifies an externally computed wall heat transfer coefficient to compute the wall heat flux. Both of these inputs have to be specified as coupled "variables".

- `name_comp_evap` and `name_comp_cond`

  If the evaporator/condenser boundary condition type is "Coupled", it accepts the name of the component coupled to the evaporator(condenser) surface of this HeatPipe component. The coupled component has to be either a PBVolumeBranch or a PBOneDFluidComponent.

- `eos_evap` and `eos_cond`

  Only when the evaporator/condenser boundary condition type is “Coupled”, and the coupled component is of PBVolumeBranch type, it is required to specify the name of equation of state to the evaporator/condenser coupled PBVolumeBranch component.

- `D_heated_evap` and `D_heated_cond`

  Only when the evaporator/condenser surface boundary condition type is “Coupled”, and the coupled component is of PBOneDFluidComponent type, it specifies the characteristic length to compute wall heat transfer coefficient in the coupled evaporator/condenser PBOneDFluidComponent component. If not specified, the coupled PBOneDFluidComponent uses its hydraulic diameter as the characteristic length for heat transfer.

- `HTC_geometry_type_evap` and `HTC_geometry_type_cond`

  Heat transfer geometry type at the evaporator/condenser surface. Acceptable options are “Pipe (default)”, “Bundle”, “Vertical-Plate”, and “Horizontal-Plate”.

- `HT_surface_area_density_evap` and `HT_surface_area_density_cond`

  Only when the evaporator/condenser surface boundary condition type is “Coupled”, and the coupled component is of PBOneDFluidComponent type, it specifies the heat transfer surface area density of this HeatPipe component with respect to the coupled evaporator/condenser surface PBOneDFluidComponent component.

- `heat_transfer_area_evap` and `heat_transfer_area_cond`

  Only when the evaporator/condenser boundary condition type is “Coupled”, and the coupled component is of PBVolumeBranch type, it specifies the evaporator/condenser side heat transfer surface area of this HeatPipe component. Since the HeatPipe component assumes normal cylinder geometry, the heated perimeter will be the perimeter of the cylinder.

- `gap_surface_name_evap` and `gap_surface_name_cond`

  When the evaporator/condenser surface boundary condition type is “GapHeatTransfer”, these parameters specify the name of the surface that is coupled with the evaporator/condenser surface through gap heat transfer.

- `h_gap_evap` and `h_surface_name_cond`

  When the evaporator/condenser surface boundary condition type is “GapHeatTransfer”, these parameters specify the constant thermal conductance in the gap.

- `length_evap`, `length_cond`, `width_evap`, and `width_cond`. When the evaporator/condenser surface boundary condition type is “GapHeatTransfer”, these parameters specify the geometry parameter for the gap. The code will calculate the thermal conductance based on the gap geometry.

An example of defining a heat pipe component is listed as below:

```
[./hp]
  type                 = HeatPipe
  position             = '0 0 0'
  orientation          = '0 0 1'
  hs_type              = cylinder
  length              = 1
  zone_lengths        = '0.3 0.4 0.3'
  width_of_hs         = '0.04 0.005 0.005'
  elem_number_radial  = '2 4 4'
  elem_numbers_axial  = '6 8 6'
  dim_hs              = 2
  material_hs         = 'vapor-mat wick-mat ss-mat'
  Ts_init             = 628.15
  HP_BC_type          = 'Convective Temperature'
  qs_evap             = 1.0e5                    # Given heat flux at evaporator surface
  T_bc_cond           = 628.15                   # Fixed temperature at condenser surface
[../]
```

4.3.39 MultiComponentArray

The MultiComponentArray component is used to create an array of `identical` components in a compact way based on the reference position vectors. It currently support components of type HeatPipe, PBOneDFluidComponent, PBPipe, and PBCoupledHeatStructure. The associated input parameters are shown below.

```
[./<MultiComponentArray>]
  component_type                 = (required)      # type of component
  ref_parameter                  = (required)      # reference input parameters for component
                                                   # array.
  positions                      = (no_default)    # Position of the individual component.
  positions_file                 = (no_default)    # A filename that should be looked in
                                                   # for positions. Each set of 3 values
                                                   # in that file will represent a Point.
                                                   # This and 'positions' cannot be both
                                                   # supplied.
  hp_cond_wall_coupled_solid_array   = (no_default)  # Name of the coupled solid array
                                                     # for the heat pipe condenser wall.
                                                     # Coupled through gap heat transfer.
  gap_surface_name_cond              = (no_default)  # Name of coupled solid array surface
                                                     # for the heat pipe condenser wall.
  hp_evap_wall_coupled_solid_array   = (no_default)  # Name of the coupled solid array for
                                                     # the heat pipe evaporator wall.
                                                     # Coupled through gap heat transfer.
  gap_surface_name_evap              = (no_default)  # Name of coupled solid array surface
                                                     # for the heat pipe evaporator wall.
  hp_cond_wall_coupled_fluid_array   = (no_default)  # Name of the coupled fluid array for
                                                     # the heat pipe condenser wall. Coupled
                                                     # through convective heat transfer.
  hp_evap_wall_coupled_fluid_array   = (no_default)  # Name of the coupled fluid array for
                                                     # the heat pipe evaporator wall. Coupled
                                                     # through convective heat transfer.
  solid_left_wall_coupled_fluid_array  = (no_default)  # Name of the coupled fluid array for
                                                       # the left wall of heat structure.
                                                       # Coupled through convective
                                                       # heat transfer.
  solid_right_wall_coupled_fluid_array = (no_default)  # Name of the coupled fluid array for
                                                       # the right wall of heat structure.
                                                       # Coupled through convective heat
                                                       # transfer.
  enable_input_helper                  = 1             # Option to print names of all blocks,
                                                       # surfaces, inlets, and outlets.

  rotation                             = 0               # Not used.
  input_parameters                     = (no_default)    # Not used.
  orientation                          = '0 0 1'         # Not used.
  position                             = '0 0 0'         # Not used.
```

```
[../]
```

The input parameters are explained below:

- `component_type` `(required)`

  Specify the type of component in this array. It currently support one of HeatPipe, PBOneDFluidComponent, PBPipe, and PBCoupledHeatStructure.

- `ref_parameter` `(required)`

  Specify the reference input parameters for the identical components in this array. The user defines the reference input parameters in the ComponentInputParameters block.

- `positions` and `positions_file`

  One of these 2 parameters should be supplied to provide the positions of individual components in this array. `positions` and `positions_file` cannot be both supplied. `positions` should be provided with a list of three-dimensional points. `positions_file` should be provided with a filename that contains the list of three-dimensional points. The total number of individual components in this array will be determined by the number of valid points provided by one of these 2 parameters.

  The individual components in this array are named as: ArrayName:c1, ArrayName:c2, etc. 'ArrayName' is the name of this array.

- `hp_cond_wall_coupled_solid_array` and `gap_surface_name_cond`

  These 2 optional input parameters work together to specify the coupling between an array of HeatPipe and an array of solid PBCoupledHeatStructure. The coupling is through gap heat transfer between the heat pipe condenser surface and a user-specified surface name of the PBCoupledHeatStructure. `hp_cond_wall_coupled_solid_array` specifies the name of the solid PBCouledHeatStructure array and `gap_surface_name_cond` specifies which surface (e.g. inner_wall and outer_wall) of PBCouledStructure will be coupled with the heat pipe condenser surface.

- `hp_evap_wall_coupled_solid_array` and `gap_surface_name_evap`

  These 2 optional input parameters work together to specify the coupling between an array of HeatPipe and an array of solid PBCoupledHeatStructure. The coupling is through gap heat transfer between the heat pipe evaporator surface and a user-specified surface name of the PBCoupledHeatStructure. `hp_evap_wall_coupled_solid_array` specifies the name of the solid PBCouledHeatStructure array and `gap_surface_name_evap` specifies which surface (e.g. inner_wall and outer_wall) of PBCouledStructure will be coupled with the heat pipe evaporator surface.

- `hp_cond_wall_coupled_fluid_array` and `hp_evap_wall_coupled_fluid_array`

  These 2 optional input parameters are used to specify the coupling between an array of HeatPipe and an array of fluid components (e.g PBOneDFluidComponent). The coupling between the heat pipe evaporator/condenser surface with the fluid component is through the convective heat transfer. `hp_cond_wall_coupled_fluid_array` and `hp_evap_wall_coupled_fluid_array` should be the name of the array of fluid coupled at the condenser surface and evaporator surface, respectively.

- solid_left_wall_coupled_fluid_array and solid_right_wall_coupled_fluid_array

  These 2 optional input parameters are used to specify the coupling between an array of PBCouled-HeatStructure and an array of fluid components (e.g PBOneDFluidComponent). The coupling between the solid left/right wall with the fluid component is through the convective heat transfer. solid_left_wall_coupled_fluid_array and solid_right_wall_coupled_fluid_array should be the name of the array of fluid coupled at the left wall (i.e. inner surface) and right wall (i.e. outer surface), respectively.

- enable_input_helper

  This optional input parameter allows the code print names of blocks, surfaces, inlets, and outlets in the output (e.g. console print). It is likely that these names will be used in the other blocks of the input model, e.g. boundary conditions and postprocessors.

An example of using the MultiComponentArray is listed as below:

```
[ComponentInputParameters]
  #-----------------------------------------------------------------------------
  # Specify parameters for the reference heat pipe in an array
  # The evaporator surface will be supplied with a heat flux, while
  # the condenser surface will be coupled with fluid through convective
  # heat transfer.
  #-----------------------------------------------------------------------------
  [./reference_hp]
    type               = HeatPipeParameters
    orientation        = '0 0 1'
    hs_type            = cylinder
    length             = 2.0
    zone_lengths       = '1.3 0.4 0.3'
    width_of_hs        = '0.015 0.001 0.001'
    elem_number_radial = '2 2 2'
    elem_numbers_axial = '26 8 6'
    dim_hs             = 2
    material_hs        = 'vapor-mat wick-mat wall-mat'
    Ts_init            = 750.0

    HP_BC_type                   = 'Convective Coupled'
    qs_evap                      = 1.0e5
    name_comp_cond               = Pipe:c1
    HT_surface_area_density_cond = 340.0
    eos_cond                     = eos
  [../]
  #-----------------------------------------------------------------------------
  # Specify parameters for the reference fluid pipe in an array
  # Notice that the fluid is only coupled with the condenser surface.
  #-----------------------------------------------------------------------------
  [./reference_pipe]
    type        = PBOneDFluidComponentParameters
    orientation = '0 0 1'
    eos         = eos
    heat_source = 0
    f           = 0.01
    length      = 0.3
    A           = 3.141591654e-4
    Dh          = 0.02
    n_elems     = 6
  [../]
[]

[Components]
```

```
  [./HP]
    type                              = MultiComponentArray
    component_type                    = HeatPipe
    positions                         = '0 0 0 0 1 0'
    ref_parameter                     = reference_hp
    hp_cond_wall_coupled_fluid_array  = Pipe
  [../]
  [./Pipe]
    type            = MultiComponentArray
    component_type  = PBOneDFluidComponent
    positions       = '0 0 1.7 0 1 1.7'
    ref_parameter   = reference_pipe
  [../]
[]
```

4.4 Control System Components

4.4.1 CTGeneric

CTGeneric is the base class for the control and trip system. It implement the computation logic of the control and trip components. The associated input parameters are shown below.

```
[./CTGeneric]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
[../]
```

Each of the input parameters are discussed as follows.

- input_names (required)

  A vector of input names to this control/trip component. A constant value can be provided as an input.

- input_types (required)

  A vector of input types to this control/trip component. The order and number of the input tyes should be consistent with that of input names. Available types are: Constant, ScalarVariable, Function, Postprocessor, ControlSystem, and TripSystem.

- start_time

  The start time of this control/trip component. If this parameter is provided, there is no actual calculation before this time and the initial value of the control/trip component is the output; otherwise, the calculation starts immediately after the creation of the control/trip component.

- `trigger_trip_name`

  The name of the triggering trip to this component. If this input parameter is provided, the computation of this component will be governed by the status of the triggering trip. If the triggering trip is true, the computation of this component is enabled; otherwise, the output of this component stays at its old value or initial value if this component has never been calculated.

- `initial_output_value`

  Initial value of this control component. In practice, when the control/trip systems are complicated, it is difficult to set the accurate initial value; in this case, users may choose not providing the initial value. This may affect accuracy of computation in the control component, but the effect is expected to dimish over time.

- `actuator`

  Option to set this control/trip component as an actuator. The actuator component will initiate the evaluation sequence in a control/trip system. Thus, at least 1 component (usually the most downstream one) needs to be set as the acutator in a control/trip system. In practice, output of an actuator component will be used in the fluid system.

- `add_output_pps`

  Option to add a postprocessor for obtaining value of this control/trip component. The added postprocess is named as ”name:value”, where ”name” is name of the control/trip component.

- `add_output_function`

  Option to add a function for obtaining value of this control/trip component. The added function is named as ”name:fn”, where ”name” is name of the control/trip component.

- `add_output_variable`

  Option to add a scalar aux variable for obtaining value of this control/trip component. The added varaible is named as ”name:out”, where ”name” is name of the control/trip component.

4.4.2 ControlSystem

In SAM, `ControlSystem` is directly inherited from `CTGeneric`, with the concept to model a control component. Its input parameters are therefore a superset of input parameters of `CTGeneric`. `ControlSystem` is base class of different type of control components. The associated input parameters are shown below.

```
[./ControlSystem]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
```

```
                                            # value of this control/trip component.
scaling_factor        = 1                   # Constant scaling factor.
[../]
```

In addition to the input parameters discussed in 4.4.1, there is one additional input parameter:

- `scaling_factor`

  A constant factor to scale value of the control component.

4.4.3 TripSystem

In SAM, `TripSystem` is directly inherited from `CTGeneric`, with the concept to model a trip component. Its input parameters are therefore a superset of input parameters of `CTGeneric`. `TripSystem` is base class of different type of trip components. The associated input parameters are shown below.

```
[./TripSystem]
input_names           = (required)   # The name of inputs to this control/trip component.
input_types           = (required)   # The type of inputs to this control/trip component.
start_time            = (no_default) # Start time of this control/trip component.
trigger_trip_name     = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                     # In most cases, this is not required.
actuator              = 0            # Option to set this control/trip component as an actuator.
add_output_pps        = 0            # Option to add a pps (name:value) to get value
                                     # of this control/trip component.
add_output_function   = 0            # Option to add a function (name:fn) to get value
                                     # of this control/trip component.
add_output_variable   = 0            # Option to add an aux scalar variable (name:out) to get
                                     # value of this control/trip component.
latched               = 0            # Whether this trip component is latched.
[../]
```

In addition to the input parameters discussed in 4.4.1, there is one additional input parameter:

- `latched`

  A boolean option for deciding if this trip is latched or unlatched. If this trip is latched, once set true, its status and TIMEOF will not change; otherwise, its status and TIMEOF will be evaluated and advanced every time step.

4.4.4 CSAddition

`CSAddition` is directly inherited from `ControlSystem` to calculate summation of input variables,

$$Y = S(A_0 + \sum_{i=1}^{N} A_i V_i) \tag{4.3}$$

The associated input parameters are shown below.

```
[./CSAddition]
input_names           = (required)   # The name of inputs to this control/trip component.
input_types           = (required)   # The type of inputs to this control/trip component.
start_time            = (no_default) # Start time of this control/trip component.
trigger_trip_name     = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                     # In most cases, this is not required.
actuator              = 0            # Option to set this control/trip component as an actuator.
```

```
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
bias                 = 0            # A constant bias to the summation.
gains                = (no_default) # Gain factors for individual input variables.
[../]
```

In addition to the input parameters discussed in 4.4.2, there are two additional input parameters:

- `bias`

  A constant bias to the summation. It corresponds to A_0 in Eq. 4.3. If not provided, a default value of 0.0 is used.

- `gains`

  A vector of gain factor for individual input variables. It corresponds to A_i in Eq. 4.3. The size of the vector should equal the number of the input variables. If not provided, a default vector of 1.0 is used.

4.4.5 CSDivision

`CSDivision` is directly inherited from `ControlSystem` to calculate division of input variables,

$$Y = \frac{S}{V_1} \text{ or } S\frac{V_2}{V_1} \tag{4.4}$$

It support either one or two input variable(s). If one input variable is provided, its inverse is calculated. The associated input parameters are shown below.

```
[./CSDivision]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
[../]
```

Other than the input parameters discussed in 4.4.2, there is no additional input parameter.

4.4.6 CSMultiplication

`CSMultiplication` is directly inherited from `ControlSystem` to calculate product of input variables,

$$Y = S\prod_{i=1}^{N} V_i \tag{4.5}$$

The associated input parameters are shown below.

```
[./CSMultiplication]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
[../]
```

Other than the input parameters discussed in 4.4.2, there is no additional input parameter.

4.4.7 CSExponentiation

`CSExponentiation` is directly inherited from `ControlSystem` to calculate power of input variables,

$$Y = SV_1^x \text{ or } SV_1^{V_2} \tag{4.6}$$

where x is exponent to the power function. It support either one or two input variable(s). If one input variable is provided, user needs to provide the exponent to the power function. If two input variables are specified, the second input is taken as exponent to the power function. The associated input parameters are shown below.

```
[./CSExponentiation]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
exponent             = (no_default) # Exponent to the power function.
[../]
```

In addition to the input parameters discussed in 4.4.2, there is one additional input parameter:

- `exponent`

  The exponent to the power function. This is required if only one input variable is provided.

4.4.8 CSSTDFunction

`CSSTDFunction` is directly inherited from `ControlSystem` to computes one of the standard function of its input variables,

$$Y = S\mathbf{F}(V_1, V_2, \cdots) \tag{4.7}$$

The associated input parameters are shown below.

```
[./CSSTDFunction]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
function_type        = (required)   # The type of the standard function.
loolup_function_name = (no_default) # The name of the loopup function.
[../]
```

In addition to the input parameters discussed in 4.4.2, there are two additional input parameter:

- `function_type` (required)

  The type of the function. Currently, the function type is one of SIN, COS, TAN, SINH, COSH, TANH, ABS, LOG, EXP, SQRT, MIN, and MAX. Correspondingly, $\mathbf{F}$ would be $\sin(V_1)$, $\cos(V_1)$, $\tan(V_1)$, $\sinh(V_1)$, $\cosh(V_1)$, $\tanh(V_1)$, $|V_1|$, $\ln(V_1)$, $\exp(V_1)$, $\sqrt{V_1}$, $\min(V_1, V_2, \cdots)$, and $\max(V_1, V_2, \cdots)$. Only min and max function may take multiple input variables. The function type is case sensitive.

- `loolup_function_name`

  Name of lookup function. In addition to the previously discussed standard functions, a user-provided function is also available to this control component. In this case, the user specifies the function type as LOOKUP and provides name of the lookup function.

4.4.9 CSDelay

`CSDelay` is directly inherited from `ControlSystem` to return its input variable after a deleyed time,

$$Y = SV_1(t - t_d) \tag{4.8}$$

The associated input parameters are shown below.

```
[./CSDelay]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
```

```
add_output_pps       = 0             # Option to add a pps (name:value) to get value
                                     # of this control/trip component.
add_output_function  = 0             # Option to add a function (name:fn) to get value
                                     # of this control/trip component.
add_output_variable  = 0             # Option to add an aux scalar variable (name:out) to get
                                     # value of this control/trip component.
scaling_factor       = 1             # Constant scaling factor.
delay_time           = (required)    # The input delay time.
num_pairs            = (required)    # The number of pairs of data used to store
                                     # past values of input variable
[../]
```

In addition to the input parameters discussed in 4.4.2, there are two additional input parameter:

- `delay_time` (required)

  The delay time to the input variable.

- `num_pairs` (required)

  The maximum number of pairs of history values to store in this component. History of the input variable is stored in form of (time, value) pair. The deleyed function is obtained by a liner interpolation of the stored history value. The ratio of dealy time to the number of pairs is better to be close to the time step size.

4.4.10 CSUnitTrip

`CSUnitTrip` is directly inherited from `ControlSystem`. It returns 0.0 or 1.0 depending on the input trip status,

$$Y = SU(\pm t_r) \tag{4.9}$$

The associated input parameters are shown below.

```
[./CSUnitTrip]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
input_complement     = 0            # Use complement of the input trip
[../]
```

In addition to the input parameters discussed in 4.4.2, there is one additional input parameter:

- `input_complement`

  Option to use complement of the input trip.

4.4.11 CSDifferentiation

CSDifferentiation is directly inherited from ControlSystem to compute time derivative of its input variable,

$$Y = S\frac{\mathrm{d}V_1}{\mathrm{d}t} \tag{4.10}$$

The associated input parameters are shown below.

```
[./CSDifferentiation]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
[../]
```

Other than the input parameters discussed in 4.4.2, there is no additional input parameter.

4.4.12 CSIntegration

CSIntegration is directly inherited from ControlSystem to compute time integral of its input variable,

$$Y = S\int_{t_0}^{t} V_1 \mathrm{d}t \tag{4.11}$$

where t_0 is the simulation time when the component is added to the system. The associated input parameters are shown below.

```
[./CSIntegration]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
[../]
```

Other than the input parameters discussed in 4.4.2, there is no additional input parameter.

4.4.13 CSProportionIntegration

`CSProportionIntegration` is directly inherited from `ControlSystem` to compute proportional-integral of its input variable,

$$Y = S\left(A_1 V_1 + A_2 \int_{t_0}^{t} V_1 \mathrm{d}t\right) \tag{4.12}$$

where t_0 is the simulation time when the component is added to the system. The associated input parameters are shown below.

```
[./CSProportionIntegration]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
scaling_factor       = 1            # Constant scaling factor.
proportion_gain      = (required)   # Gain factor for the proportion part.
integration_gain     = (required)   # Gain factor for the integration part.
[../]
```

In addition to the input parameters discussed in 4.4.2, there are two additional input parameters:

- `proportion_gain` (required)

  The gain factor for the proportion part. It corresponds to A_1 in Eq. 4.12.

- `integration_gain` (required) The gain factor for the integration part. It corresponds to A_2 in Eq. 4.12.

4.4.14 CSLeadLag

`CSLeadLag` is directly inherited from `ControlSystem` to compute lead-lad operation of its input variable,

$$Y(s) = S\left(\frac{1+A_1 s}{1+A_2 s}\right) V_1(s) \tag{4.13}$$

where A_1 and A_2 are the lead and lag time constant, respectively. The associated input parameters are shown below.

```
[./CSLeadLag]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
```

```
                                         # of this control/trip component.
add_output_variable   = 0                # Option to add an aux scalar variable (name:out) to get
                                         # value of this control/trip component.
scaling_factor        = 1                # Constant scaling factor.
lead_time_constant    = (required)       # Lead time constant.
lag_time_constant     = (required)       # Lag time constant.
[../]
```

In addition to the input parameters discussed in 4.4.2, there are two additional input parameters:

- lead_time_constant (required)

  The lead time constant. It corresponds to A_1 in Eq. 4.13. It can be set to zero, which is equivalent to a pure lag operation.

- lag_time_constant (required) The lag time constant. It corresponds to A_2 in Eq. 4.13. It can be set to zero, which is equivalent to a pure lead operation.

4.4.15 TSCompare

TSCompare is directly inherited from TripSystem to perform the comparison of its input variables,

$$T_r = V_1 \mathbf{OP} (V_2 + C) \tag{4.14}$$

where C is a user-specified constant bias and OP is the comparison operator. This component support either one or two input variables. If only one input variable is provided, the first variable is compared to the constant bias. The associated input parameters are shown below.

```
[./TSCompare]
input_names           = (required)   # The name of inputs to this control/trip component.
input_types           = (required)   # The type of inputs to this control/trip component.
start_time            = (no_default) # Start time of this control/trip component.
trigger_trip_name     = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                     # In most cases, this is not required.
actuator              = 0            # Option to set this control/trip component as an actuator.
add_output_pps        = 0            # Option to add a pps (name:value) to get value
                                     # of this control/trip component.
add_output_function   = 0            # Option to add a function (name:fn) to get value
                                     # of this control/trip component.
add_output_variable   = 0            # Option to add an aux scalar variable (name:out) to get
                                     # value of this control/trip component.
latched               = 0            # Whether this trip component is latched.
bias                  = 0            # The bias to the second input.
compare_type          = (required)   # The type of the comparison operation.
[../]
```

In addition to the input parameters discussed in 4.4.3, there are two additional input parameter:

- bias

  A constant bias to the second input. If not provided, a default value of 0.0 is used.

- compare_type (required)

  The type of the comparison. It is one of the following relational operations: EQ (equal), NE (not equal), GT (greater than), GE (greater then or equal), LT (less than), and LE (less than or equal).

4.4.16 TSBoolean

`TSBoolean` is directly inherited from `TripSystem` to perform the boolean operation of its input trip signals,

$$T_r = T_{r,1}\mathbf{OP}T_{r,2} \tag{4.15}$$

where OP is the boolean operator. The associated input parameters are shown below.

```
[./TSBoolean]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
latched              = 0            # Whether this trip component is latched.
boolean_type         = 0            # The type of boolean operation.
[../]
```

In addition to the input parameters discussed in 4.4.1, there is one additional input parameter:

- `boolean_type`

  The type of the boolean operation. It is one of the logical operations: AND, OR (inclusive or), NAND (not-and), NOR (not-ro), XOR (exclusive or), and XNOR (exclusive NOR). An extra operation (NOT) is also available if only 1 input is specified for taking the complement of the input trip.

4.4.17 TSDelay

`TSBoolean` is directly inherited from `TripSystem` to return the input trip signal at a delayed time,

$$T_r = T_{r,1}(t - t_d) \tag{4.16}$$

where $T_{r,1}$ is the input trip signal, t is the current time, and t_d is the delay time.

```
[./TSDelay]
input_names          = (required)   # The name of inputs to this control/trip component.
input_types          = (required)   # The type of inputs to this control/trip component.
start_time           = (no_default) # Start time of this control/trip component.
trigger_trip_name    = (no_default) # Name of the triggering trip of this control/trip component.
initial_output_value = (no_default) # Initial output value of this control/trip component.
                                    # In most cases, this is not required.
actuator             = 0            # Option to set this control/trip component as an actuator.
add_output_pps       = 0            # Option to add a pps (name:value) to get value
                                    # of this control/trip component.
add_output_function  = 0            # Option to add a function (name:fn) to get value
                                    # of this control/trip component.
add_output_variable  = 0            # Option to add an aux scalar variable (name:out) to get
                                    # value of this control/trip component.
latched              = 0            # Whether this trip component is latched.
boolean_type         = 0            # The type of boolean operation.
```

```
delay_time            = 0            # The delay time.
[../]
```

In addition to the input parameters discussed in 4.4.1, there is one additional input parameter:

- delay_time

  The delay time of the input trip. The output signal is set to true if the input signal is true and $t - t_d$ is larger than TIMEOF value of the input trip.

4.4.18 Input syntax

Because all control/trip components are based on the GeneralUserObject of MOOSE, inputs of control/trip components lie in the UserObjects block. A user can add control/trip components using the syntax for adding a normal UserObject. An example is given as follows :

```
[Functions]
  [./inputFn]
    type = ParsedFunction
    value = 't'
  [../]
[]
[UserObjects]
  [./C1]                        # pi*t
    type = CSAddition           # This is a CSAddition type of control component.
    input_names = 'inputFn'     # A function as the input to the control component.
    input_types = 'Function'    # The input is of Function type.
    gains = '3.14'              # The gain factor to the input.
  [../]
  [./C2]                        # S * sin(pi*t)
    type = CSSTDFunction        # This is a CSSTDFunction type of control component.
    input_names = 'C1'          # The control component 'C1' is taken as the input.
    input_types = 'ControlSystem' # The input is of a ControlSystem type.
    function_type = 'SIN'       # The function type is sine function.
    add_output_pps = true       # Add a pps to print the value.
    scaling_factor = 1.5        # Scale the value by 1.5.
  [../]
  [./T1]                        # S * sin(pi*t) > 1.0 ?
    type = TSCompare            # This is a TSCompare type of trip component.
    input_names = 'C2 1.0'      # The input is a control component and a constant value
    input_types = 'ControlSystem Constant' # The input types are ControlSystem and Constant
    compare_type = 'GT'         # The comparison type is GT (greater than).
    add_output_pps = true       # Add a pps to print the TIMEOF value.
  [../]
[]
```

In practice, there could be a large amount of simple control/trip components in an input model. A special class, CTInputBlock, is added to help add a series of control/trip components in a more compact way. CTInputBlock is aimed to condense the required information for adding multiple control/trip components.

CTInputBlock is inherited from GeneralUserObject. The main function of this class is to process the input parameters to a series of control/trip components and add these components internally. The associated input parameters are shown below.

```
[./CTInputBlock]
ct_name               = (required)   # The name of all components to add.
ct_type               = (required)   # The type of all components to add.
ct_input_names        = (required)   # The parameter 'input_names' of all components to add.
ct_input_types        = (required)   # The parameter 'input_types' of all components to add.
```

```
add_output_pps        = (no_default) # The parameter 'add_output_pps' of certain components.
add_output_function   = (no_default) # The parameter 'add_output_funtion' of certain components.
add_output_variable   = (no_default) # The parameter 'add_output_variable' of certain components.
start_time            = (no_default) # The parameter 'start time' of certain component.
trigger_trip_name     = (no_default) # The parameter 'trigger_trip_name' of certain component.
initial_output_value = (no_default) # The parameter 'initial_output_value'' of certain components.
scaling_factor        = (no_default) # The parameter 'scaling_factor' of certain components.
latched               = (no_default) # The parameter 'latched' of certain components.
bias                  = (no_default) # The parameter 'bias' of certain components.
gains                 = (no_default) # The parameter 'gains' of certain components.
lead_time_constant    = (no_default) # The parameter 'lead_time_constant' of certain components.
lag_time_constant     = (no_default) # The parameter 'lag_time_constant' of certain components.
delay_time            = (no_default) # The parameter 'delay_time' of certain components.
num_pairs             = (no_default) # The parameter 'num_pairs' of certain components.
exponent              = (no_default) # The parameter 'exponent' of certain components.
proportion_gain       = (no_default) # The parameter 'proportion_gain' of certain components.
integration_gain      = (no_default) # The parameter 'integration_gain' of certain components.
function_type         = (no_default) # The parameter 'function_type' of certain components.
lookup_function_name = (no_default) # The parameter 'lookup_function_name' of certain components.
input_complement      = (no_default) # The parameter 'input_complement' of certain components.
compare_type          = (no_default) # The parameter 'compare_type' of certain components.
boolean_type          = (no_default) # The parameter 'boolean_type' of certain components.
[../]
```

Input parameters of the CTInputBlock are union of input parameters of all different types of control/trip components. The meaning of these input parameters were thus already discussed in previous sections. The input parameters and its format are discussed belows.

- ct_name (required)

  This input parameter is a vector names (i.e. string). It specifies the name of control/trip components to add in this block. The different names are separated by whitespace, e.g.

  ```
          ct_name = 'C1 C2 T1'    # 3 control/trip components will be added in this block
  ```

- ct_type (required)

  This input parameter is a vector types (i.e. string). It specifies the type of control/trip components to add in this block. The different types are separated by whitespace, e.g.

  ```
          ct_type = 'CSAddition CSSTDFunction TSCompare'
  ```

- ct_input_names (required)

  This input parameter is a vector of input_names to the individual control/trip component to add in this block. Because input_names is itself a vector of string, ct_input_names is thus an array of string. The differnt rows (i.e. individual input_names) are separated by semicolon(;). For example,

  ```
          ct_input_names = 'inputFn; C1; C2 1.0'
  ```

- ct_input_types (required)

  This input parameter is a vector of input_types to the individual control/trip component to add in this block. Because input_types is itself a vector of string, ct_input_types is thus an array of string. The differnt rows (i.e. individual input_types) are separated by semicolon(;). For example,

```
        ct_input_types = 'Function; ControlSystem; ControlSystem Constant'
```

- optional input parameters

  The remaining optional input parameters follow the same format. Each input parameter is itself a single formated string. The string is in the format of (name, value) pairs, where 'name' is simply the name of the control/trip component that needs to specify this input parameter. The different pairs are separated by semicolon(;). For example,

```
        gains          = 'C1 3.14'            # Only C1 needs the gains input
        add_output_pps = 'C2 true; T1 true'   # C2 and T1 will add a pps
        scaling_factor = 'C2 1.5'             # Only C2 needs the scaling_factor input
        compare_type   = 'T1 GT'              # Only T1 needs the compare_type input
        function_type  = 'C2 SIN'             # Only C2 needs the function type
```

Making use of the CTInputBlock, the previous example 4.4.18 can be condensed to

```
[Functions]
  [./inputFn]
    type = ParsedFunction
    value = 't'
  [../]
[]
[UserObjects]
  [./block0]
    type = CTInputBlock
    ct_name        = 'C1 C2 T1'
    ct_type        = 'CSAddition CSSTDFunction TSCompare'
    ct_input_names = 'inputFn; C1; C2 1.0'
    ct_input_types = 'Function; ControlSystem; ControlSystem Constant'
    gains          = 'C1 3.14'
    add_output_pps = 'C2 true; T1 true'
    scaling_factor = 'C2 1.5'
    compare_type   = 'T1 GT'
    function_type  = 'C2 SIN'
  [../]
[]
```

4.5 ComponentInputParameters

Independent to the [Components] input block, SAM also provides a separate [ComponentInput-Parameters] input block, where users could provide input template for certain types of SAM components. Note that this input block only provides component 'template', and by itself, no real components will created. There are two common usages of this input block: 1) to provide the common features of a type of component, which will be used as reference to build real components in the [Components] input block; and 2) to provide a completely predefined component that will be referred to and created as a sub-component of a composite-type of component. The two common usages are to be discussed in the remaining part of this section. Currently, the SAM components that supports such a feature is listed in Table 4.1.

The first usage is to provide common features for components input. When preparing SAM input files to perform thermal-hydraulics analysis, it is typical to observe that many components share common features, for example, a test loop with the majority of it built from a type of standard

Table 4.1: SAM components that supports ComponentInputParameters feature

ComponentInputParameters	**SAM Component Name**
DuctedFuelAssemblyParameters	DuctedFuelAssembly
HeatPipeParameters	HeatPipe
HeatStructureParameters	HeatStructure
MultiChannelRodBundleParameters	MultiChannelRodBundle
PBCoreChannelParameters	PBCoreChannel
PBOneDFluidComponentParameters	PBOneDFluidComponent
PBPipeParameters	PBPipe

ASME pipe. In this case, `[ComponentInputParameters]` can be used to provide the abstracted common features of such a type of components, and `[Components]` only provides component-specific parameters and refers to these common features to generate the complete input parameter list. With this approach, it greatly reduces users' burden to type in the same parameters for many times, and also reduces the possibility of input error. Note that input parameters specified in the `[Components]` input blocks can override what has been provided as common features provided in the `[ComponentInputParameters]` input block. An example is given as follows:

```
[ComponentInputParameters]
  # This sub-block provides component input parameter with common features
  [./Schedule-10-w-insulation]
    type = PBPipeParameters
    eos = eos
    A = 6.097763E-04
    heat_source = 0
    Dh = 2.786380E-02
    hs_type = cylinder
    Twall_init = 2.981500E+02
    heat_source_solid = '0 0'
    dim_wall = 2
    wall_thickness = '0.0027686  0.0508'
    n_wall_elems = '2  4'
    material_wall = 'SS-304  Fiberglass'
    HS_BC_type = Temperature
    T_wall = 2.981500E+02
    HT_surface_area_density = 355.5              # This parameter will be overridden
                                                 # in one of the two components
  [../]
[]

[Components]
  [./pipe-1]
    type = PBPipe
    input_parameters = Schedule-10-w-insulation   # This refers to the PBPipeParameters
                                                  # with common features
    length = 1
    position = '0 0 0'
    orientation = '1 0 0'
    n_elems = 20
    initial_V = -0.1
  [../]

  [./pipe-2]
    type = PBPipe
    input_parameters = Schedule-10-w-insulation   # This refers to the PBPipeParameters
```

```
                                                   # with common features
    length = 2
    position = '1 0 0'
    orientation = '0 0 1'
    n_elems = 25
    initial_V = -0.1
    HT_surface_area_density = 100.0                # This parameter overrides what has been
                                                   # provided in Schedule-10-w-insulation
  [../]
[]
```

The other usage is to provide needed input parameters, which predefines a component that will be referred to and created as a sub-component of a composite-type of component. Currently, this only happens to a special component, `HexLatticeCore`, which relies on several predefined components to create its sub-components (see section 4.3.11 for more details). An example is given as follows:

```
[ComponentInputParameters]
  [./F1]  # Predefined PBCoreChannel
    type = PBCoreChannelParameters
    eos = eos
    A = 0.005105685
    Dh = 0.003446961
    length = 0.8
    n_elems = 20
    HT_surface_area_density = 1068.182718
    dim_hs = 2
    name_of_hs = 'fuel gap clad'
    Ts_init = 628.15
    n_heatstruct = 3
    fuel_type = cylinder
    width_of_hs = '0.003015 0.000465  0.00052'
    elem_number_of_hs = '5 1 1'
    material_hs = 'fuel-mat gap-mat clad-mat'
    power_shape_function = ppf_axial
  [../]

  [./reference_hs]  # Predefined heat structures for intra-assembly duct walls and gap
    type = HeatStructureParameters
    hs_type = plate
    length = 0.8
    dim_hs = 2
    elem_number_axial = 20
    elem_number_radial = '2  2  2'
    width_of_hs = '0.003 0.004 0.003'
    material_hs = 'duct-mat gap-mat duct-mat'
    hs_names = 'duct_i gap duct_o'
    Ts_init = 628.15
    HS_BC_type = 'Coupled Coupled'
    HT_surface_area_density_left = 15.3766
    HT_surface_area_density_right = 15.3766
  [../]

  [./duct_wall]  # Predefined heat structures for duct wall
    type = HeatStructureParameters
    hs_type = plate
    length = 0.8
    dim_hs = 2
    elem_number_axial = 20
    elem_number_radial = 2
    width_of_hs = '0.003'
    material_hs = 'duct-mat'
    hs_names = 'duct'
    Ts_init = 628.15
    HS_BC_type = 'Coupled Adiabatic'
    HT_surface_area_density_left = 15.3766
  [../]
[]

[Components]
  [./core]
    type = HexLatticeCore
    position = '0 0 0'
    orientation = '0 0 1'
    n_side = 2
    assem_pitch = 0.14598
    assem_Dft = 0.13598
    radial_power_peaking = '1 1 1 1.5 1 1 0.5'

    assem_layout = 'F1 F1 F1 F1 F1 F1 F1'  # F1 refers to the predefined
                                           # PBCoreChannelParameters
    ref_hs = reference_hs       # This refers to a predefined heat structure
    ref_duct = duct_wall        # This refers to a predefined duct wall heat structure
  [../]
[]
```

4.6 PostProcessors

4.6.1 ComponentBoundaryEnergyBalance

This Postprocessor is designed to monitor the energy flux balance between two selected pipe ends. A common usage is to monitor the energy balance of a pipe component on its two ends, and compare it with the total heat source applied to this pipe.

```
[./ComponentBoundaryEnergyBalance]
  eos         = (required)       # The name of equation of state object to use.
  execute_on  = TIMESTEP_END     # The list of flag(s) indicating when this object should
                                 # be executed, the available options include NONE, INITIAL,
                                 # LINEAR, NONLINEAR, TIMESTEP_END,
                                 # TIMESTEP_BEGIN, FINAL, CUSTOM.
  input       = (required)       # Name of the components and boundaries
[../]
```

- `input` `(required)`

  This input parameter specifies a list of two pipe ends, where energy fluxes are to be compared to compute an energy balance between them:

  $$(\rho u h A)_2 - (\rho u h A)_1 \,.$$

  The input syntax is similar to those for junction type of component, e.g., `input = ‘pipe-1(in) pipe-1(out)’` or `input = ‘IHX(secondary_in) IHX(secondary_out)’`.

- `eos` `(required)`

  Equation of state used in the pipe component.

- `execute_on`

  This is an input parameter inherited from MOOSE framework, it specifies how often this Postprocessor should perform a computation. It is common to all Postprocessors to be discussed in this section, and in general, it is safe to not specify anything.

4.6.2 ComponentBoundaryFlow

This Postprocessor is simply monitors the mass flow rate, $\rho u A$, of a pipe end.

```
[./ComponentBoundaryFlow]
  input        = (required)  # Name of the components and boundaries
  scale_factor = 1           # Scale factor to be applied to the ordinate values
[../]
```

- `input` (required)

  This input parameter specifies a pipe and one of its ends, where mass flow rate is to be computed as $\rho u A$. The input syntax is similar to those for junction type of component, e.g., `input = pump_pipe(in)`.

- `scale_factor`

  This is a scaling factor to be multiplied to the mass flow rate. The default value is 1.

4.6.3 ComponentBoundaryScalarFlow

This Postprocessor is similar to ComponentBoundaryFlow, and it simply monitors the flow rate of a passive scalar, $\rho u A \phi$, of a pipe end.

```
[./ComponentBoundaryScalarFlow]
  input        = (required)  # Name of the components and boundaries
  variable     = (required)  # Name of the particle
  scale_factor = 1           # Scale factor to be applied to the ordinate values
[../]
```

- `input` (required)

  This input parameter specifies a pipe and one of its ends, where the flow rate of a passive scalar is to be computed as $\rho u A \phi$. The input syntax is similar to those for junction type of component, e.g., `input = pump_pipe(in)`.

- `variable` (required)

  The name of the passive scalar variable.

- `scale_factor`

  This is a scaling factor to be multiplied to the flow rate of the passive scalar. The default value is 1.

4.6.4 ComponentBoundaryVariableValue

This Postprocessor returns the value of a specified variable at a pipe end.

```
[./ComponentBoundaryVariableValue]
  input        = (required)  # Name of the components and boundaries
  variable     = (required)  # Name of the variable
  scale_factor = 1           # Scale factor to be applied to the ordinate values
[../]
```

- input (required)

  The same as in ComponentBoundaryScalarFlow.

- variable (required)

  The name of the variable, such as “pressure”, “temperature”, “velocity”, “rho” (fluid density), “enthalpy”, “heat_transfer_coefficient” (if modeled), and passive scalars (if modeled).

- scale_factor

  This is a scaling factor to be multiplied to the variable value. The default value is 1.

4.6.5 ComponentNodalVariableValue

This Postprocessor returns the value of a specified variable on a specified node in a pipe.

```
[./ComponentNodalVariableValue]
  input        = (required)  # Name of the components and boundaries
  variable     = (required)  # Name of the variable
  scale_factor = 1           # Scale factor to be applied to the ordinate values
[../]
```

- input (required)

  This input parameter specifies a pipe and a node id, where the value of the specified variable will be returned. The input syntax is, for example, `input = pipe(0)` or `input = IHX:primary_pipe(10)`. Note that node id starts from 0.

- variable (required)

  The same as in ComponentBoundaryVariableValue.

- scale_factor

  This is a scaling factor to be multiplied to the variable value. The default value is 1.

4.6.6 ConductionHeatRemovalRate

This Postprocessor computes the integral heat removal rate from a side of a two-dimensional heat structure.

```
[./ConductionHeatRemovalRate]
  boundary          = (required)  # The list of boundary IDs from
                                  # the mesh where this boundary
                                  # condition applies
  heated_perimeter = (required)  # The length of the HeatExchanger heated perimeter
[../]
```

- boundary (required)

  This input parameter specifies the boundary name where the integral heat removal rate to be computed, for example, `boundary = ‘hp0:cond_wall’`.

- heated_perimeter (required)

  The heated perimeter of the boundary to compute the integral heat removal rate,

$$Q = \int -k\nabla T P_h dL$$

  in which, $-k\nabla T$ is the local surface heat flux, dL is the length along the boundary side, and P_h is this heated perimeter input parameter.

4.6.7 HeatExchangerHeatRemovalRate

This Postprocessor computes the integral heat removal rate from the wall heat structure of a heat exchanger to a specified pipe, e.g., the primary or the secondary side pipe.

```
[./HeatExchangerHeatRemovalRate]
  block            = (required)  # The list of block ids (SubdomainID)
                                 # that this object will be applied
  heated_perimeter = (required)  # The length of the HeatExchanger heated perimeter
[../]
```

- block (required)

  This input parameter specifies the block name where the integral heat removal rate to be computed, for example, block = 'DHX:primary_pipe'.

- heated_perimeter (required)

  The heated perimeter of the boundary to compute the integral heat removal rate,

$$Q = \int h\left(T_f - T_{wall}\right) P_h dL$$

  in which, h is the local heat transfer coefficient, T_f is the local fluid temperature, T_{wall} is the local wall surface temperature, dL is the length along the pipe, and P_h is this heated perimeter input parameter.

4.7 TimeSteppers

4.7.1 CourantNumberTimeStepper

The CourantNumberTimeStepper is a TimeStepper inherited from PostprocessorDT, which computes time step size based on a Postprocessor value, in this case, MaxCourantNumber. Its input parameters are listed as follows:

```
[./<CourantNumberTimeStepper>]
  Courant_number  = 10                                     # Target Courant number
  dt              = (no_default)                           # Initial value of dt
  factor          = 0                                      # Add a factor to the
                                                           # supplied postprocessor
                                                           # value.
  postprocessor   = Simulation:MaxCourantNumber(required)  # The name of the postprocessor
                                                           # that computes the dt
  reset_dt        = 0                                      # Use when restarting
                                                           # a calculation to force
                                                           # a change in dt.
  scale           = 1                                     # Multiple scale and
                                                          # supplied postprocessor
                                                          # value.
  type            = CourantNumberTimeStepper
[../]
```

Input parameters are discussed as follows:

- `Courant_number`

  This TimeStepper adjusts the time step size to match this given Courant number as a user input parameter. The default value is 10.

- `dt`

  The initial value of time step size for this TimeStepper to start with. If not specified, the code uses a default value of 0.01 second.

- `factor` and `scale`

  These two input parameters are not used.

- `postprocessor`

  You do NOT and should NOT specify this input parameter. A default value, `Simulation: MaxCourantNumber`, has been automatically generated and given to this parameter.

- `reset_dt (advanced MOOSE option)`

  Use when restarting a calculation to force a change in dt. By default, it is false (0).

An example input of the `CourantNumberTimeStepper` block is shown below.

```
[./TimeStepper]
  type = CourantNumberTimeStepper
  dt = 0.02
  Courant_number = 0.5
[../]
```

This input block should be used as an sub-block of the `Executioner` input block.

4.8 Preconditioning

The Preconditioning block describes the preconditioner to be used by the preconditioned JFNK solver (available through PETSc). Two options are currently available, the single matrix preconditioner (SMP) and the finite difference preconditioner (FDP). The FDP option uses numerical Jacobian by doing direct finite differences of the residual terms. It is normally slow, and only intended for debugging purposes. The SMP option is more efficient and the recommended option. The input parameters of the Preconditioning block are shown below. An example input block follows.

```
[Preconditioning]
  [./*]
    active                          = __all__  # If specified only the blocks named will be
                                               # visited and made active
    line_search                     = default  # Specifies the line search type (Note:
                                               # none = basic)
    petsc_options                   =          # Singleton PETSc options
    petsc_options_iname             =          # Names of PETSc name/value pairs
    petsc_options_value          =           # Values of PETSc name/value pairs (must
                                             # correspond with "petsc_options_iname"
    solve_type                   =           # PJFNK: Preconditioned Jacobian-Free Newton
                                             # Krylov JFNK, NEWTON, FD, LINEAR
  [../]

  [./FDP]
    control_tags                 =           # Adds user-defined labels for accessing
                                             # object parameters via control logic.
    enable                       = 1         # Set the enabled status of the MooseObject.
    full                         = 0         # Set to true if you want the full set of
                                             # couplings.
    implicit_geometric_coupling  = 0         # Set to true if you want to add entries into
                                             # the matrix for degrees of freedom that might
                                             # be coupled by inspection of the geometric
                                             # search objects.
    line_search                  = default   # Specifies the line search type (Note:
                                             # none = basic)
    off_diag_column              =           # The off diagonal column you want to add into
                                             # the matrix, it will be associated with an
                                             # off diagonal row from the same position in
                                             # off_diag_row.
    off_diag_row                 =           # The off diagonal row you want to add into
                                             # the matrix, it will be associated
                                             # with an off diagonal column from the same
                                             # position in off_diag_colum.
    pc_side                      = right     # Preconditioning side
    petsc_options                =           # Singleton PETSc options
    petsc_options_iname          =           # Names of PETSc name/value pairs
    petsc_options_value          =           # Values of PETSc name/value pairs (must
                                             # correspond with "petsc_options_iname"
    solve_type                   =           # PJFNK: Preconditioned Jacobian-Free Newton
                                             # Krylov JFNK, NEWTON, FD, LINEAR
    type                         = FDP
  [../]

  [./SMP]
    control_tags        =                    # Adds user-defined labels for accessing
                                             # object parameters via control logic.
    coupled_groups      =                    # List multiple space separated groups of
                                             # comma separated variables. Off-diagonal
                                             # jacobians will be generated for all pairs
                                             # within a group.
    enable              = 1                  # Set the enabled status of the MooseObject.
    full                = 0                  # Set to true if you want the full set of
                                             # couplings.
    line_search         = default            # Specifies the line search type (Note:
                                             # none = basic)
    off_diag_column     =                    # The off diagonal column you want to add into
                                             # the matrix, it will be associated with an
                                             # off diagonal row from the same position in
                                             # off_diag_row.
    off_diag_row        =                    # The off diagonal row you want to add into
                                             # the matrix, it will be associated
                                             # with an off diagonal column from the same
                                             # position in off_diag_colum.
    pc_side             = right              # Preconditioning side
    petsc_options       =                    # Singleton PETSc options
    petsc_options_iname =                    # Names of PETSc name/value pairs
    petsc_options_value =                    # Values of PETSc name/value pairs (must
                                             # correspond with "petsc_options_iname"
    solve_type          =                    # PJFNK: Preconditioned Jacobian-Free Newton
                                             # Krylov JFNK, NEWTON, FD, LINEAR
    type                = SMP
  [../]
[]
```

```
[Preconditioning]
   active = 'SMP_PJFNK'

  [./SMP_PJFNK]
    type = SMP                          # Single-Matrix Preconditioner
    full = true                         # Using the full set of couplings among all variables
    solve_type = 'PJFNK'                # Using Preconditioned JFNK solution method
    petsc_options_iname = '-pc_type'    # Names of PETSc name/value pairs
    petsc_options_value = 'lu'          # Values of PETSc name/value pairs
  [../]

[]
```

4.9 Executioner

The Executioner block describes the calculation process flow used in the simulation. The common MOOSE Executioners are also listed here, and the associated input parameters of the Executioner block are shown below. An example of the Executioner input block is also followed. Common SAM MOOSE Executioner types include: CoupledCFDExecutioner (for coupled simulation with CFD codes), CoupledSASTransient (for coupled SAS/SAM transient simulations), Steady (for steady state simulation), and Transient (for transient simulations).

```
[./<Executioner>]
[Executioner]
  active                         = __all__    # If specified only the blocks named will be
                                              # visited and made active

  petsc_options_iname            =            # petsc options names
  petsc_options_value            =            # petsc options values
  scheme                         = bdf2       # Time integration scheme used.

  [./<type>]
    [./CoupledCFDExecutioner]
      CFD_scaling_factor         = 1          # the scaling factor in the CFD model
      SYSCFDBoundaryConsistency  = (required) # if the SYS and CFD Boundaries are
                                              # consistent
      abort_on_solve_fail        = 0          # abort if solve not converged rather than
                                              # cut timestep
      compute_initial_residual_before_preset_bcs = 0 # Use the residual norm computed
                                              # *before* PresetBCs are imposed in relative
                                              # convergence check
      control_tags               =            # Adds user-defined labels for accessing
                                              # object parameters via control logic.
      dt                         = 1          # The timestep size between solves
      dtmax                      = 1e+30    # The maximum timestep size in an adaptive run
      dtmin                      = 2e-14    # The minimum timestep size in an adaptive run
      enable                     = 1          # Set the enabled status of the MooseObject.
      end_time                   = 1e+30      # The end time of the simulation
      input_data_file            = (required) # Input data file from external coupling
      isRestarting               = 0          # if it is a restart coupled code simulation
      l_abs_step_tol             = -1         # Linear Absolute Step Tolerance
      l_max_its                  = 10000      # Max Linear Iterations
      l_tol                      = 1e-05      # Linear Tolerance
      line_search                = default    # Specifies the line search type
                                              # (Note: none = basic)
      n_in_parameter             = (required) # Number of coupling input parameters
      n_out_parameter            = (required) # Number of coupling output parameters
      n_startup_steps            = 0          # The number of timesteps during startup
      name_of_in_components       = (required) # Names of coupling input components
      name_of_in_parameters       = (required) # Parameter names of coupling input
                                               # components
      name_of_out_components      = (required) # Names of coupling output components
      name_of_out_parameters      = (required) # Variable names of coupling output
                                               # components
      names_of_CFD_boundary       = (required) # names of coupled CFD boundaries
      nl_abs_step_tol             = 1e-50      # Nonlinear Absolute step Tolerance
      nl_abs_tol                  = 1e-50      # Nonlinear Absolute Tolerance
      nl_max_funcs                = 10000      # Max Nonlinear solver function evaluations
      nl_max_its                  = 50         # Max Nonlinear Iterations
      nl_rel_step_tol             = 1e-50      # Nonlinear Relative step Tolerance
      nl_rel_tol                  = 1e-08      # Nonlinear Relative Tolerance
      no_fe_reinit                = 0          # Specifies whether or not to reinitialize
                                               # FEs
      num_steps                   = 4294967295 # The number of timesteps in a transient run
      output_data_file            = (required) # Output data file for external coupling
      petsc_options               =            # Singleton PETSc options
      petsc_options_iname         =            # Names of PETSc name/value pairs
      petsc_options_value         =            # Values of PETSc name/value pairs (must
                                               # correspond with "petsc_options_iname"
      picard_abs_tol              = 1e-50      # The absolute nonlinear residual to shoot
                                               # for during Picard iterations. This check is
                                               # performed based on the Master app's
                                               # nonlinear residual.
      picard_max_its              = 1          # Number of times each timestep will be
                                               # solved. Mainly used when wanting to do
                                               # Picard iterations with MultiApps that
                                               # are set to execute_on
                                               # timestep_end or timestep_begin
      picard_rel_tol              = 1e-08      # The relative nonlinear residual drop
                                               # to shoot for during Picard iterations.
                                               # This check is performed based on the Master
                                               # app's nonlinear residual.
      reset_dt                    = 0          # Use when restarting a calculation to force
                                               # a change in dt.
      restart_file_base           =            # File base name used for restart
      scheme                      =            # Time integration scheme used.
      solve_type                  =            # PJFNK: Preconditioned Jacobian-Free Newton
                                               # Krylov JFNK, NEWTON, FD, LINEAR
      splitting                   =            # Top-level splitting defining a hierarchical
                                               # decomposition into subsystems to help
                                               # the solver.
      ss_check_tol                = 1e-08      # Whenever the relative residual changes by
                                               # less than this the solution
                                               # will be considered to be at steady state.
      ss_tmin                     = 0          # Minimum number of timesteps to take before
                                               # checking for steady state conditions.
      start_time                  = 0          # The start time of the simulation
      time_period_ends            =            # The end times of time periods
      time_period_starts          =            # The start times of time periods
      time_periods                =            # The names of periods
      timestep_tolerance          = 2e-14      # the tolerance setting for final timestep
                                               # size and sync times
      trans_ss_check              = 0          # Whether or not to check for steady state
                                               # conditions
      type                        = CoupledCFDExecutioner
      use_multiapp_dt             = 0          # If true then the dt for the simulation will
                                               # be chosen by the MultiApps. If false
                                               # (the default) then the minimum over the
                                               # master dt and the MultiApps is used
      verbose                     = 0          # Print detailed diagnostics on timestep
                                               # calculation
    [../]
```

```
[./CoupledSASTransient]
  abort_on_solve_fail        = 0           # abort if solve not converged rather than
                                           # cut timestep
  compute_initial_residual_before_preset_bcs = 0 # Use the residual norm computed
                                           # *before* PresetBCs are imposed in relative
                                           # convergence check
  control_tags               =             # Adds user-defined labels for accessing
                                           # object parameters via control logic.
  coupling_components        = (required) # Names of coupling components
  dt                         = 1           # The timestep size between solves
  dtmax                      = 1e+30     # The maximum timestep size in an adaptive run
  dtmin                      = 2e-14     # The minimum timestep size in an adaptive run
  enable                     = 1           # Set the enabled status of the MooseObject.
  end_time                   = 1e+30       # The end time of the simulation
  input_fifo                 = (required) # Input data named pipe from external
                                           # coupling
  l_abs_step_tol             = -1          # Linear Absolute Step Tolerance
  l_max_its                  = 10000       # Max Linear Iterations
  l_tol                      = 1e-05       # Linear Tolerance
  line_search                = default     # Specifies the line search type
                                           # (Note: none = basic)
  n_startup_steps            = 0           # The number of timesteps during startup
  nl_abs_step_tol            = 1e-50       # Nonlinear Absolute step Tolerance
  nl_abs_tol                 = 1e-50       # Nonlinear Absolute Tolerance
  nl_max_funcs               = 10000       # Max Nonlinear solver function evaluations
  nl_max_its                 = 50          # Max Nonlinear Iterations
  nl_rel_step_tol            = 1e-50       # Nonlinear Relative step Tolerance
  nl_rel_tol                 = 1e-08       # Nonlinear Relative Tolerance
  no_fe_reinit               = 0           # Specifies whether or not to reinitialize
                                           # FEs
  num_steps                  = 4294967295 # The number of timesteps in a transient run
  output_fifo                = (required) # Output data named pipe from external
                                           # coupling
  petsc_options              =             # Singleton PETSc options
  petsc_options_iname        =             # Names of PETSc name/value pairs
  petsc_options_value        =             # Values of PETSc name/value pairs (must
                                           # correspond with "petsc_options_iname"
  picard_abs_tol             = 1e-50       # The absolute nonlinear residual to shoot
                                           # for during Picard iterations. This check is
                                           # performed based on the Master app's
                                           # nonlinear residual.
  picard_max_its             = 1           # Number of times each timestep will be
                                           # solved. Mainly used when wanting to do
                                           # Picard iterations with MultiApps that
                                           # are set to execute_on
                                           # timestep_end or timestep_begin
  picard_rel_tol             = 1e-08       # The relative nonlinear residual drop
                                           # to shoot for during Picard iterations.
                                           # This check is performed based on the Master
                                           # app's nonlinear residual.
  reset_dt                   = 0           # Use when restarting a calculation to force
                                           # a change in dt.
  restart_file_base          =             # File base name used for restart
  scheme                     =             # Time integration scheme used.
  solve_type                 =             # PJFNK: Preconditioned Jacobian-Free Newton
                                           # Krylov JFNK, NEWTON, FD, LINEAR
  splitting                  =             # Top-level splitting defining a hierarchical
                                           # decomposition into subsystems to help
                                           # the solver.
  ss_check_tol               = 1e-08       # Whenever the relative residual changes by
                                           # less than this the solution
                                           # will be considered to be at steady state.
  ss_tmin                    = 0           # Minimum number of timesteps to take before
```

```
                                          # checking for steady state conditions.
    start_time                 = 0        # The start time of the simulation
    time_period_ends           =          # The end times of time periods
    time_period_starts         =          # The start times of time periods
    time_periods               =          # The names of periods
    timestep_tolerance         = 2e-14    # the tolerance setting for final timestep
                                          # size and sync times
    trans_ss_check             = 0        # Whether or not to check for steady state
                                          # conditions
    type                       = CoupledSASTransient
    use_multiapp_dt            = 0        # If true then the dt for the simulation will
                                          # be chosen by the MultiApps. If false
                                          # (the default) then the minimum over the
                                          # master dt and the MultiApps is used
    verbose                    = 0        # Print detailed diagnostics on timestep
                                          # calculation
  [../]

  [./Steady]
    compute_initial_residual_before_preset_bcs = 0
                                          # Use the residual norm computed *before*
                                          # PresetBCs are imposed in relative
                                          # convergence check
    control_tags               =          # Adds user-defined labels for accessing
                                          # object parameters via control logic.
    enable                     = 1        # Set the enabled status of the MooseObject.
    l_abs_step_tol             = -1       # Linear Absolute Step Tolerance
    l_max_its                  = 10000    # Max Linear Iterations
    l_tol                      = 1e-05    # Linear Tolerance
    line_search                = default  # Specifies the line search type
                                          # (Note: none = basic)
    nl_abs_step_tol            = 1e-50    # Nonlinear Absolute step Tolerance
    nl_abs_tol                 = 1e-50    # Nonlinear Absolute Tolerance
    nl_max_funcs               = 10000    # Max Nonlinear solver function evaluations
    nl_max_its                 = 50       # Max Nonlinear Iterations
    nl_rel_step_tol            = 1e-50    # Nonlinear Relative step Tolerance
    nl_rel_tol                 = 1e-08    # Nonlinear Relative Tolerance
    no_fe_reinit               = 0        # Specifies whether or not to reinitialize
                                          # FEs
    petsc_options              =          # Singleton PETSc options
    petsc_options_iname        =          # Names of PETSc name/value pairs
    petsc_options_value        =          # Values of PETSc name/value pairs (must
                                          # correspond with "petsc_options_iname"
    restart_file_base          =          # File base name used for restart
    solve_type                 =          # PJFNK: Preconditioned Jacobian-Free Newton
                                          # Krylov JFNK, NEWTON, FD, LINEAR
    splitting                  =          # Top-level splitting defining a hierarchical
                                          # decomposition into subsystems to help
                                          # the solver.
    type                       = Steady
  [../]

  [./Transient]
    abort_on_solve_fail        = 0        # abort if solve not converged rather than
                                          # cut timestep
    control_tags               =          # Adds user-defined labels for accessing
                                          # object parameters via control logic.
    dt                         = 1        # The timestep size between solves
    dtmax                      = 1e+30  # The maximum timestep size in an adaptive run
    dtmin                      = 2e-14  # The minimum timestep size in an adaptive run
    enable                     = 1        # Set the enabled status of the MooseObject.
    end_time                   = 1e+30    # The end time of the simulation

    l_abs_step_tol             = -1       # Linear Absolute Step Tolerance
    l_max_its                  = 10000      # Max Linear Iterations
    l_tol                      = 1e-05      # Linear Tolerance
    line_search                = default    # Specifies the line search type
                                            # (Note: none = basic)
    n_startup_steps            = 0          # The number of timesteps during startup
    nl_abs_step_tol            = 1e-50      # Nonlinear Absolute step Tolerance
    nl_abs_tol                 = 1e-50      # Nonlinear Absolute Tolerance
    nl_max_funcs               = 10000      # Max Nonlinear solver function evaluations
    nl_max_its                 = 50         # Max Nonlinear Iterations
    nl_rel_step_tol            = 1e-50      # Nonlinear Relative step Tolerance
    nl_rel_tol                 = 1e-08      # Nonlinear Relative Tolerance
    no_fe_reinit               = 0          # Specifies whether or not to reinitialize
                                            # FEs
    num_steps                  = 4294967295 # The number of timesteps in a transient run
    petsc_options              =            # Singleton PETSc options
    petsc_options_iname        =            # Names of PETSc name/value pairs
    petsc_options_value        =            # Values of PETSc name/value pairs (must
                                            # correspond with "petsc_options_iname"
    picard_abs_tol             = 1e-50      # The absolute nonlinear residual to shoot
                                            # for during Picard iterations. This check is
                                            # performed based on the Master app's
                                            # nonlinear residual.
    picard_max_its             = 1          # Number of times each timestep will be
                                            # solved. Mainly used when wanting to do
                                            # Picard iterations with MultiApps that
                                            # are set to execute_on
                                            # timestep_end or timestep_begin
    picard_rel_tol             = 1e-08      # The relative nonlinear residual drop
                                            # to shoot for during Picard iterations.
                                            # This check is performed based on the Master
                                            # app's nonlinear residual.
    reset_dt                   = 0          # Use when restarting a calculation to force
                                            # a change in dt.
    restart_file_base          =            # File base name used for restart
    scheme                     =            # Time integration scheme used.
    solve_type                 =            # PJFNK: Preconditioned Jacobian-Free Newton
                                            # Krylov JFNK, NEWTON, FD, LINEAR
    splitting                  =            # Top-level splitting defining a hierarchical
                                            # decomposition into subsystems to help
                                            # the solver.
    ss_check_tol               = 1e-08      # Whenever the relative residual changes by
                                            # less than this the solution
                                            # will be considered to be at steady state.
    ss_tmin                    = 0          # Minimum number of timesteps to take before
                                            # checking for steady state conditions.
    start_time                 = 0          # The start time of the simulation
    time_period_ends           =            # The end times of time periods
    time_period_starts         =            # The start times of time periods
    time_periods               =            # The names of periods
    timestep_tolerance         = 2e-14      # the tolerance setting for final timestep
                                            # size and sync times
    trans_ss_check             = 0          # Whether or not to check for steady state
                                            # conditions
    type                       = Transient
    use_multiapp_dt            = 0          # If true then the dt for the simulation will
                                            # be chosen by the MultiApps. If false
                                            # (the default) then the minimum over the
                                            # master dt and the MultiApps is used
    verbose                    = 0          # Print detailed diagnostics on timestep
                                            # calculation
  [../]

[./Quadrature]
  active                       = __all__    # If specified only the blocks named will be
```

```
                                                 # visited and made active
    element_order                  = AUTO        # Order of the quadrature for elements
    order                          = AUTO        # Order of the quadrature
    side_order                     = AUTO        # Order of the quadrature for sides
    type                           = GAUSS       # Type of the quadrature rule
  [../]

  [./TimeStepper]
    active                         = __all__     # If specified only the blocks named will be
                                                 # visited and made active

    [./<type>]
      [./ContinueOnDtMinTimeStepper]
        control_tags               =             # Adds user-defined labels for accessing
                                                 # objectparameters via control logic.
        enable                     = 1           # Set the enabled status of the MooseObject.
        growth_factor              = 2           # Maximum ratio of new to previous timestep
                                                 # sizes following a step that required the
                                                 # time step to be cut due to a failed solve.
        interpolate                = 1           # Whether or not to interpolate DT between
                                                 # times. This is true by default for
                                                 # historical reasons.
        min_dt                     = 0           # The minimal dt to take.
        reset_dt                   = 0           # Use when restarting a calculation to force
                                                 # a change in dt.
        time_dt                    =             # The values of dt
        time_t                     =             # The values of t
        type                       = ContinueOnDtMinTimeStepper
      [../]

      [./CourantNumberTimeStepper]
        Courant_number             = 10          # Target Courant number
        control_tags               =             # Adds user-defined labels for accessing
                                                 # object parameters via control logic.
        dt                         =             # Initial value of dt
        enable                     = 1           # Set the enabled status of the MooseObject.
        postprocessor              = Simulation:MaxCourantNumber
                                                 # The name of the postprocessor that
                                                 # computes the dt
        reset_dt                   = 0           # Use when restarting a calculation to force
                                                 # a change in dt.
        type                       = CourantNumberTimeStepper
      [../]

      [./FunctionDT]
        control_tags               =             # Adds user-defined labels for accessing
                                                 # object parameters via control logic.
        enable                     = 1           # Set the enabled status of the MooseObject.
        growth_factor              = 2           # Maximum ratio of new to previous timestep
                                                 # sizes following a step that required the
                                                 # time step to be cut due to a failed solve.
        interpolate                = 1           # Whether or not to interpolate DT between
                                                 # times. This is true by default for
                                                 # historical reasons.
        min_dt                     = 0           # The minimal dt to take.
        reset_dt                   = 0           # Use when restarting a calculation to force
                                                 # a change in dt.
        time_dt                    =             # The values of dt
        time_t                     =             # The values of t
        type                       = FunctionDT
      [../]

      [./PostprocessorDT]
        control_tags               =             # Adds user-defined labels for accessing
                                                 # object parameters via control logic.
        dt                     =              # Initial value of dt
        enable                 = 1            # Set the enabled status of the MooseObject.
        postprocessor          = (required)   # The name of the postprocessor that computes
                                              # the dt
        reset_dt               = 0            # Use when restarting a calculation to force
                                              # a change in dt.
        type                   = PostprocessorDT
      [../]
    [../]
  [../]
[]

[../]
```

```
[Executioner]
  type = Transient                      # This is a transient simulation

  dt = 1e-1                             # Targeted time step size
  dtmin = 1e-10                         # The allowed minimum time step size

  petsc_options_iname = '-ksp_gmres_restart'  # Additional PETSc settings, name list
  petsc_options_value = '300'                 # Additional PETSc settings, value list

  nl_rel_tol = 1e-7                     # Relative nonlinear tolerance for each Newton solve
  nl_abs_tol = 1e-6                     # Relative nonlinear tolerance for each Newton solve
  nl_max_its = 30                     # Number of nonlinear iterations for each Newton solve

  l_tol = 1e-6                          # Relative linear tolerance for each Krylov solve
  l_max_its = 100                       # Number of linear iterations for each Krylov solve

  start_time = 0.0                      # Physical time at the beginning of the simulation
  num_steps = 100                       # Max. simulation time steps
  end_time = 100.                       # Max. physical time at the end of the simulation

  [./Quadrature]
    type = TRAP                         # Using trapezoid integration rule
    order = FIRST                       # Order of the quadrature
  [../]
[]
```

4.10 Outputs

The Outputs block specifies various settings of different output types (screen display and files) in the simulation. The input parameters of common MOOSE Outputs are shown below, with an example Outputs block followed. Common MOOSE output types include:

- CSV: write post-processor and scalar variables to a separate comma-separated-values file,
- Checkpoint: save snapshots of the simulation data including all meshes, solutions, and stateful object data,
- Console: output to screen with runtime information,
- Exocdus: write all mesh and solution data to an ExodusII file.

```
[Outputs]
  active                       = __all__    # If specified only the blocks named will be
```

```
                                              # visited and made active
  [./CSV]
    additional_execute_on        =            # This list of output flags is added to the
                                              # existing flags  (initial|linear|nonlinear|
                                              # timestep_end|timestep_begin|final|
                                              # failed|custom)
                                              # to execute only at that moment
    align                        = 0          # Align the outputted csv data by padding
                                              # the numbers with trailing whitespace
    append_date                  = 0          # When true the date and time are appended
                                              # to the output filename.
    append_date_format           =            # The format of the date/time to append,
                                              # if not given UTC format used (see
                                              # http://www.cplusplus.com/reference
                                              # /ctime/strftime).
    append_restart               = 0          # Append existing file on restart
    control_tags                 =            # Adds user-defined labels for accessing
                                              # object parameters via control logic.
    delimiter                    =            # Assign the delimiter (default is ','
    enable                       = 1          # Set the enabled status of the MooseObject.
    end_time                     =            # Time at which this output object stop
                                              # operating
    execute_elemental_variables  = 1          # Enable/disable the output of elemental
                                              # variables
    execute_input                = 1          # Enable/disable the output of input file
                                              # information
    execute_nodal_variables      = 1          # Enable/disable the output of nodal
                                              # variables
    execute_on                   = 'INITIAL TIMESTEP_END'      # Set to
                                              # (none|initial|linear|nonlinear|
                                              # timestep_end|timestep_begin|final|
                                              # failed|custom)
                                              # to execute only at that moment
    execute_postprocessors_on    =            # Control of when postprocessors are output
    execute_scalar_variables     = 1          # Enable/disable the output of aux scalar
                                              # variables
    execute_scalars_on           =            # Control the output of scalar variables
    execute_system_information   = 1          # Enable/disable the output of the simulation
                                              # information
    execute_vector_postprocessors = 1         # Enable/disable the output of vector
                                              # postprocessors
    execute_vector_postprocessors_on =        # Enable/disable the output of
                                              #VectorPostprocessors
    file_base                    =            # The desired solution output name without an
                                              # extension
    hide                         =            # A list of the variables and postprocessors
                                              # that should NOT be output to the Exodus
                                              # file (may include Variables,
                                              # ScalarVariables, and Postprocessor names).
    interval                     = 1          # The interval at which time steps are output
                                              # to the solution file
    linear_residual_dt_divisor   = 1000       # Number of divisions applied to time step
                                              # when outputting linear residuals
    linear_residual_end_time     =            # Specifies an end time to begin output on
                                              # each linear residual evaluation
    linear_residual_start_time   =            # Specifies a start time to begin output on
                                              # each linear residual evaluation
    linear_residuals             = 0          # Specifies whether output occurs on each
                                              # linear residual evaluation
    nonlinear_residual_dt_divisor = 1000      # Number of divisions applied to time step
                                              # when outputting non-linear residuals
    nonlinear_residual_end_time  =            # Specifies an end time to begin output on
                                              # each nonlinear residual evaluation
    nonlinear_residual_start_time =           # Specifies a start time to begin output on
```

```
                                         # each nonlinear residual evaluation
    nonlinear_residuals          = 0     # Specifies whether output occurs on each
                                         # nonlinear residual evaluation
    output_if_base_contains      =       # If this is supplied then output will only
                                         # be done in the case that the output base
                                         # contains one of these strings. This is
                                         # helpful in outputting only a subset of
                                         # outputs when using MultiApps.
    output_linear                = 0     # Specifies whether output occurs on each
                                         # linear residual evaluation
    output_nonlinear             = 0     # Specifies whether output occurs on each
                                         # nonlinear residual evaluation
    output_postprocessors        = 1     # Enable/disable the output of postprocessors
    precision                    = 14    # Set the output precision
    show                         =       # A list of the variables and postprocessors
                                         # that should be output to the Exodus file
                                         # (may include Variables, ScalarVariables,
                                         # and Postprocessor names).
    start_time                   =       # Time at which this output object begins to
                                         # operate
    sync_only                    = 0     # Only export results at sync times
    sync_times                   =       # Times at which the output and solution is
                                         # forced to occur
    time_data                    = 0     # When true and VecptorPostprocessor data
                                         # exists, write a csv file containing
                                         # the timestep and time information.
    time_tolerance               = 1e-14 # Time tolerance utilized checking start and
                                         # end times
    type                         = CSV
    use_displaced                = 0     # Enable/disable the use of the displaced
                                         # mesh for outputting
  [../]

  [./Checkpoint]
    additional_execute_on        =       # This list of output flags is added to the
                                         # existing flags  (initial|linear|nonlinear|
                                         # timestep_end|timestep_begin|final|
                                         # failed|custom)
                                         # to execute only at that moment
    append_date                  = 0     # When true the date and time are appended
                                         # to the output filename.
    append_date_format           =       # The format of the date/time to append,
                                         # if not given UTC format used (see
                                         # http://www.cplusplus.com/reference
                                         # /ctime/strftime).
    binary                       = 1     # Toggle the output of binary files
    control_tags                 =       # Adds user-defined labels for accessing
                                         # object parameters via control logic.
    enable                       = 1     # Set the enabled status of the MooseObject.
    end_time                     =       # Time at which this output object stop
                                         # operating
    execute_on                   = 'INITIAL TIMESTEP_END'      # Set to
                                         # (none|initial|linear|nonlinear|
                                         # timestep_end|timestep_begin|final|
                                         # failed|custom)
                                         # to execute only at that moment
    file_base                    =       # The desired solution output name without an
                                         # extension
    interval                     = 1     # The interval at which time steps are output
                                         # to the solution file
    linear_residual_dt_divisor   = 1000  # Number of divisions applied to time step
                                         # when outputting linear residuals
    linear_residual_end_time     =       # Specifies an end time to begin output on
                                         # each linear residual evaluation
    linear_residual_start_time   =               # Specifies a start time to begin output on
                                                 # each linear residual evaluation
    linear_residuals             = 0             # Specifies whether output occurs on each
                                                 # linear residual evaluation
    nonlinear_residual_dt_divisor = 1000         # Number of divisions applied to time step
                                                 # when outputting non-linear residuals
    nonlinear_residual_end_time  =               # Specifies an end time to begin output on
                                                 # each nonlinear residual evaluation
    nonlinear_residual_start_time =              # Specifies a start time to begin output on
                                                 # each nonlinear residual evaluation
    nonlinear_residuals          = 0             # Specifies whether output occurs on each
                                                 # nonlinear residual evaluation
    num_files                    = 2             # Number of the restart files to save
    output_if_base_contains      =               # If this is supplied then output will only
                                                 # be done in the case that the output base
                                                 # contains one of these strings. This is
                                                 # helpful in outputting only a subset of
                                                 # outputs when using MultiApps.
    output_linear                = 0             # Specifies whether output occurs on each
                                                 # linear residual evaluation
    output_nonlinear             = 0             # Specifies whether output occurs on each
                                                 # nonlinear residual evaluation
    padding                      = 4             # The number of for extension suffix (e.g.,
                                                 # out.e-s002)
    start_time                   =               # Time at which this output object begins to
                                                 # operate
    suffix                       = cp            # This will be appended to the file_base to
                                                 # create the directory name for checkpoint
                                                 # files.
    sync_only                    = 0             # Only export results at sync times
    sync_times                   =               # Times at which the output and solution is
                                                 # forced to occur
    time_data                    = 0             # When true and VecptorPostprocessor data
                                                 # exists, write a csv file containing
                                                 # the timestep and time information.
    time_tolerance               = 1e-14         # Time tolerance utilized checking start and
                                                 # end times
    type                         = Checkpoint
    use_displaced                = 0             # Enable/disable the use of the displaced
                                                 # mesh for outputting
  [../]

  [./Console]
    additional_execute_on        =               # This list of output flags is added to the
                                                 # existing flags  (initial|linear|nonlinear|
                                                 # timestep_end|timestep_begin|final|
                                                 # failed|custom)
                                                 # to execute only at that moment
    all_variable_norms           = 0             # If true, all variable norms will be printed
                                                 # after each solve
    append_date                  = 0             # When true the date and time are appended
                                                 # to the output filename.
    append_date_format           =               # The format of the date/time to append,
                                                 # if not given UTC format used (see
                                                 # http://www.cplusplus.com/reference
                                                 # /ctime/strftime).
    append_restart               = 0             # Append existing file on restart
    control_tags                 =               # Adds user-defined labels for accessing
                                                 # object parameters via control logic.
    enable                       = 1             # Set the enabled status of the MooseObject.
    end_time                     =               # Time at which this output object stop
                                                 # operating
    execute_elemental_variables  = 1             # Enable/disable the output of elemental
                                                 # variables
```

```
execute_input                 = 1          # Enable/disable the output of input file
                                           # information
execute_input_on              =            # Enable/disable the output of the input file
execute_nodal_variables       = 1          # Enable/disable the output of nodal
                                           # variables
execute_on                    = 'INITIAL TIMESTEP_END'      # Set to
                                           # (none|initial|linear|nonlinear|
                                           # timestep_end|timestep_begin|final|
                                           # failed|custom)
                                           # to execute only at that moment
execute_postprocessors_on     =            # Control of when postprocessors are output
execute_scalar_variables      = 1          # Enable/disable the output of aux scalar
                                           # variables
execute_scalars_on            =            # Control the output of scalar variables
execute_system_information    = 1          # Enable/disable the output of the simulation
                                           # information
execute_vector_postprocessors = 1          # Enable/disable the output of vector
                                           # postprocessors
execute_vector_postprocessors_on =         # Enable/disable the output of
                                           #VectorPostprocessors
file_base                     =            # The desired solution output name without an
                                           # extension
fit_mode                      = ENVIRONMENT # Specifies the wrapping mode for post
                                           # -processor tables that are printed to
                                           # the screen
                                           # (ENVIRONMENT: Read "MOOSE_PPS_WIDTH" for
                                           # desired width, AUTO: Attempt to determine
                                           # width automatically (serial only), <n>:
                                           # Desired width
hide                          =            # A list of the variables and postprocessors
                                           # that should NOT be output to the Exodus
                                           # file (may include Variables,
                                           # ScalarVariables, and Postprocessor names).
interval                      = 1          # The interval at which time steps are output
                                           # to the solution file
linear_residual_dt_divisor    = 1000       # Number of divisions applied to time step
                                           # when outputting linear residuals
linear_residual_end_time      =            # Specifies an end time to begin output on
                                           # each linear residual evaluation
linear_residual_start_time    =            # Specifies a start time to begin output on
                                           # each linear residual evaluation
linear_residuals              = 0          # Specifies whether output occurs on each
                                           # linear residual evaluation
max_rows                      = 15         # The maximum number of postprocessor/scalar
                                           # values displayed on screen
                                           # during a timestep (set to 0 for unlimited)
nonlinear_residual_dt_divisor = 1000       # Number of divisions applied to time step
                                           # when outputting non-linear residuals
nonlinear_residual_end_time   =            # Specifies an end time to begin output on
                                           # each nonlinear residual evaluation
nonlinear_residual_start_time =            # Specifies a start time to begin output on
                                           # each nonlinear residual evaluation
nonlinear_residuals           = 0          # Specifies whether output occurs on each
                                           # nonlinear residual evaluation
outlier_multiplier            = '0.8 2'    # Multiplier utilized to determine if a
                                           # residual norm is an outlier. If the
                                           # variable residual is less than
                                           # multiplier[0] times the total
                                           # residual it is colored red. If the
                                           # variable residual is
                                           # less than multiplier[1] times
                                           # the average residual it is colored yellow.
outlier_variable_norms        = 1          # If true, outlier variable norms will be
                                           # printed after each solve
```

```
    output_file                  = 0            # Output to the file
    output_if_base_contains      =              # If this is supplied then output will only
                                                # be done in the case that the output base
                                                # contains one of these strings. This is
                                                # helpful in outputting only a subset of
                                                # outputs when using MultiApps.
    output_linear                = 0            # Specifies whether output occurs on each
                                                # linear residual evaluation
    output_nonlinear             = 0            # Specifies whether output occurs on each
                                                # nonlinear residual evaluation
    output_postprocessors        = 1            # Enable/disable the output of postprocessors
    output_screen                = 1            # Output to the screen
    padding                      = 4            # The number of for extension suffix (e.g.,
                                                # out.e-s002)
    perf_header                  =              # Print the libMesh performance log header
                                                # (requires that 'perf_log = true')
    perf_log                     = 0            # If true, all performance logs will be
                                                # printed. The individual log settings will
                                                # override this option.
    perf_log_interval            = 0            # If set, the performance log will be printed
                                                # every n time steps
    print_mesh_changed_info      = 0            # When true, each time the mesh is changed
                                                # the mesh information is printed
    scientific_time              = 0            # Control the printing of time and dt in
                                                # scientific notation
    setup_log                    =              # Toggles the printing of the 'Setup
                                                # Performance' log
    setup_log_early              = 0            # Specifies whether or not the Setup
                                                # Performance log should be printed before
                                                # the first time step.  It will still be
                                                # printed at the end if perf_log
                                                # is also enabled and likewise disabled
                                                # if perf_log is false
    show                         =              # A list of the variables and postprocessors
                                                # that should be output to the Exodus file
                                                # (may include Variables, ScalarVariables,
                                                # and Postprocessor names).
    show_multiapp_name           = 0            # Indent multiapp output using the
                                                # multiapp name
    solve_log                    =              # Toggles the printing of the 'Moose Test
                                                # Performance' log
    start_time                   =              # Time at which this output object begins to
                                                # operate
    sync_only                    = 0            # Only export results at sync times
    sync_times                   =              # Times at which the output and solution is
                                                # forced to occur
    system_info                  = 'AUX EXECUTION FRAMEWORK MESH NONLINEAR'
                                                # List of information types
                                                # to display ('framework', 'mesh', 'aux',
                                                # 'nonlinear', 'execution', 'output')
    time_data                    = 0            # When true and VecptorPostprocessor data
                                                # exists, write a csv file containing
                                                # the timestep and time information.
    time_precision               =              # The number of significant digits that are
                                                # printed on time related outputs
    time_tolerance               = 1e-14        # Time tolerance utilized checking start and
                                                # end times
    type                         = Console
    use_displaced                = 0            # Enable/disable the use of the displaced
                                                # mesh for outputting
    verbose                      = 0            # Print detailed diagnostics on timestep
                                                # calculation
  [../]
```

```
[./Exodus]
  additional_execute_on        =             # This list of output flags is added to the
                                             # existing flags  (initial|linear|nonlinear|
                                             # timestep_end|timestep_begin|final|
                                             # failed|custom)
                                             # to execute only at that moment
  append_date                  = 0           # When true the date and time are appended
                                             # to the output filename.
  append_date_format           =             # The format of the date/time to append,
                                             # if not given UTC format used (see
                                             # http://www.cplusplus.com/reference
                                             # /ctime/strftime).
  append_oversample            = 0           # Append '_oversample' to the output
                                             # file base
  control_tags                 =             # Adds user-defined labels for accessing
                                             # object parameters via control logic.
  enable                       = 1           # Set the enabled status of the MooseObject.
  end_time                     =             # Time at which this output object stop
                                             # operating
  execute_elemental_on         =             # Control the output of elemental variables
  execute_elemental_variables  = 1           # Enable/disable the output of elemental
                                             # variables
  execute_input                = 1           # Enable/disable the output of input file
                                             # information
  execute_input_on             =             # Enable/disable the output of the input file
  execute_nodal_on             =             # Control the output of nodal variables
  execute_nodal_variables      = 1           # Enable/disable the output of nodal
                                             # variables
  execute_on                   = 'INITIAL TIMESTEP_END'      # Set to
                                             # (none|initial|linear|nonlinear|
                                             # timestep_end|timestep_begin|final|
                                             # failed|custom)
                                             # to execute only at that moment
  execute_postprocessors_on    =             # Control of when postprocessors are output
  execute_scalar_variables     = 1           # Enable/disable the output of aux scalar
                                             # variables
  execute_scalars_on           =             # Control the output of scalar variables
  execute_system_information   = 1           # Enable/disable the output of the simulation
                                             # information
  execute_vector_postprocessors = 1          # Enable/disable the output of vector
                                             # postprocessors
  file                         =             # The name of the mesh file to read, for
                                             # oversampling
  file_base                    =             # The desired solution output name without an
                                             # extension
  hide                         =             # A list of the variables and postprocessors
                                             # that should NOT be output to the Exodus
                                             # file (may include Variables,
                                             # ScalarVariables, and Postprocessor names).
  interval                     = 1           # The interval at which time steps are output
                                             # to the solution file
  linear_residual_dt_divisor   = 1000        # Number of divisions applied to time step
                                             # when outputting linear residuals
  linear_residual_end_time     =             # Specifies an end time to begin output on
                                             # each linear residual evaluation
  linear_residual_start_time   =             # Specifies a start time to begin output on
                                             # each linear residual evaluation
  linear_residuals             = 0           # Specifies whether output occurs on each
                                             # linear residual evaluation
  nonlinear_residual_dt_divisor = 1000       # Number of divisions applied to time step
                                             # when outputting non-linear residuals
  nonlinear_residual_end_time  =             # Specifies an end time to begin output on
                                             # each nonlinear residual evaluation
  nonlinear_residual_start_time =            # Specifies a start time to begin output on
```

```
                                             # each nonlinear residual evaluation
    nonlinear_residuals          = 0         # Specifies whether output occurs on each
                                             # nonlinear residual evaluation
    output_if_base_contains      =           # If this is supplied then output will
                                             # only be done in the case that the output
                                             # base contains one of these strings. This
                                             # is helpful in outputting only a subset of
                                             # outputs when using MultiApps.
    output_linear                = 0         # Specifies whether output occurs on each
                                             # linear residual evaluation
    output_material_properties   = 0         # Flag indicating if material properties
                                             # should be output
    output_nonlinear             = 0         # Specifies whether output occurs on each
                                             # nonlinear residual evaluation
    output_postprocessors        = 1         # Enable/disable the output of postprocessors
    oversample                   = 0         # Set to true to enable oversampling
    overwrite                    = 0         # When true the latest timestep will
                                             # overwrite the existing file, so only
                                             # a single timestep exists.
    padding                      = 3         # The number of for extension suffix (e.g.,
                                             # out.e-s002)
    position                     =           # Set a positional offset, this vector will
                                             # get added to the nodal coordinates to move
                                             # the domain.
    refinements                  = 0         # Number of uniform refinements for
                                             # oversampling (refinement levels beyond
                                             # any uniform refinements)
    scalar_as_nodal              = 0         # Output scalar variables as nodal
    sequence                     =           # Enable/disable sequential file output
                                             # (enabled by default when 'use_displace
                                             # = true', otherwise defaults to false
    show                         =           # A list of the variables and postprocessors
                                             # that should be output to the Exodus file
                                             # (may include Variables, ScalarVariables,
                                             # and Postprocessor names).
    show_material_properties     =           # List of materialproperties that should be
                                             # written to the output
    start_time                   =           # Time at which this output object begins to
                                             # operate
    sync_only                    = 0         # Only export results at sync times
    sync_times                   =           # Times at which the output and solution is
                                             # forced to occur
    time_tolerance               = 1e-14     # Time tolerance utilized checking start and
                                             # end times
    type                         = Exodus
    use_displaced                = 0         # Enable/disable the use of the displaced
                                             # mesh for outputting
    use_problem_dimension        =           # Use the problem dimension to the mesh
                                             # output. Set to false when outputting lower
                                             # dimensional meshes embedded in a higher
                                             # dimensional space.
  [../]

  [./VariableResidualNormsDebugOutput]
    additional_execute_on        =           # This list of output flags is added to the
                                             # existing flags  (initial|linear|nonlinear|
                                             # timestep_end|timestep_begin|final|
                                             # failed|custom)
                                             # to execute only at that moment
    control_tags                 =           # Adds user-defined labels for accessing
                                             # object parameters via control logic.
    delimiter                    =           # Assign the delimiter (default is ','
    enable                       = 1         # Set the enabled status of the MooseObject.
    end_time                     =           # Time at which this output object stop
```

```
                                      # operating
    execute_on                   = 'INITIAL TIMESTEP_END'     # Set to
                                      # (none|initial|linear|nonlinear|
                                      # timestep_end|timestep_begin|final|
                                      # failed|custom)
                                      # to execute only at that moment
    interval                     = 1          # The interval at which time steps are output
                                      # to the solution file
    linear_residual_dt_divisor   = 1000       # Number of divisions applied to time step
                                      # when outputting linear residuals
    linear_residual_end_time     =            # Specifies an end time to begin output on
                                      # each linear residual evaluation
    linear_residual_start_time   =            # Specifies a start time to begin output on
                                      # each linear residual evaluation
    linear_residuals             = 0          # Specifies whether output occurs on each
                                      # linear residual evaluation
    nonlinear_residual_dt_divisor = 1000      # Number of divisions applied to time step
                                      # when outputting non-linear residuals
    nonlinear_residual_end_time  =            # Specifies an end time to begin output on
                                      # each nonlinear residual evaluation
    nonlinear_residual_start_time =           # Specifies a start time to begin output on
                                      # each nonlinear residual evaluation
    nonlinear_residuals          = 0          # Specifies whether output occurs on each
                                      # nonlinear residual evaluation
    output_linear                = 0          # Specifies whether output occurs on each
                                      # linear residual evaluation
    output_nonlinear             = 0          # Specifies whether output occurs on each
                                      # nonlinear residual evaluation
    #
    start_time                   =            # Time at which this output object begins to
                                      # operate
    sync_only                    = 0          # Only export results at sync times
    sync_times                   =            # Times at which the output and solution is
                                      # forced to occur
    time_tolerance               = 1e-14      # Time tolerance utilized checking start and
                                      # end times
    type                         = VariableResidualNormsDebugOutput
    use_displaced                = 0          # Enable/disable the use of the displaced
                                      # mesh for outputting
  [../]

[]
```

```
[Outputs]
  [./checkpoint]
    type = Checkpoint                     # Save snapshots of the simulation data
  [../]
  [./console]
    type = Console                        # Screen output
    perf_log = true                       # Output the performance log
  [../]
  [./out_displaced]
    type = Exodus                         # Output simulation data to an ExodusII file
    use_displaced = true                  # Use displaced mesh
    execute_on = 'initial timestep_end'   # Output data at the begining of the simulation
                                          # and each time step
    sequence = false                      # Don't save sequential file output per
                                          # time step
  [../]
[]
```

5 Example Problems

In this section, several examples are given to demonstrate that how SAM is used to perform nuclear reactor safety related thermal-hydraulics analysis with input file also provided.

5.1 Heat Conduction Problem

The 2-D radial and axial steady-state conduction equation was solved for a generic long solid rod, as illustrated in Figure 5.1. The same case is also included in the TRACE fundamental validation cases [17]. The heat structure has a length of 20 cm and radius of 5 mm. It has a uniform heat source of 1000 W distributed within the rod, and constant thermal conductivity of 2 W/mK. The solid rod is immersed in a pool of water having a constant temperature of 300 K in the bottom 10 cm and 500 K in the top 10 cm. A constant heat transfer coefficient of 1000 W/m^{2}K is applied to the outer surface of the rod. The tabulated analytical solution values from Table A.1.2 of Reference [17] are used here in Table 5.1 for comparison to the temperatures calculated by SAM.

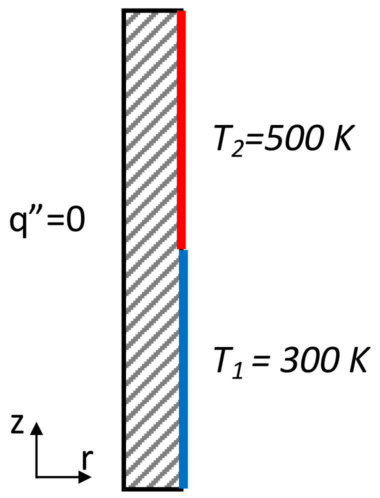


Figure 5.1: SAM model of the 2-D heat conduction problem.

Table 5.1: Comparison of SAM and Analytical Solutions for the Steady State Axial-Radial Heat Conduction Problem

Location (m)	**Analytical (K)**	**SAM (K)**	**Error (K)**
0	658.1	658.1	0.0
0.01	658.1	658.1	0.0
0.02	658.1	658.1	0.0
0.03	658.1	658.1	0.0
0.04	658.1	658.1	0.0
0.05	658.1	658.1	0.0
0.06	658.1	658.1	0.0
0.07	658.1	658.1	0.0
0.08	658.2	658.2	0.0
0.09	662.7	662.3	-0.4
0.1	758.1	758.1	0.0
0.11	853.9	853.9	0.0
0.12	857.9	858.0	0.1
0.13	858.1	858.1	0.0
0.14	858.1	858.1	0.0
0.15	858.1	858.1	0.0
0.16	858.1	858.1	0.0
0.17	858.1	858.1	0.0
0.18	858.1	858.1	0.0
0.19	858.1	858.1	0.0

The SAM simulation was run with Steady solver for this test case. The calculated steady-state conditions and analytical solution of centerline temperature distributions are compared in Table 5.1 and shown in Figure 5.2. The results given in Table 5.1 demonstrate that the SAM solutions of the 2-D heat-conduction equation are accurate. The largest errors are where temperature profile is steepening. Note that a relative coarse mesh, 40 (20-axial and 2-radial) elements total, was used in SAM simulations. The errors can be reduced if a finer mesh is used.

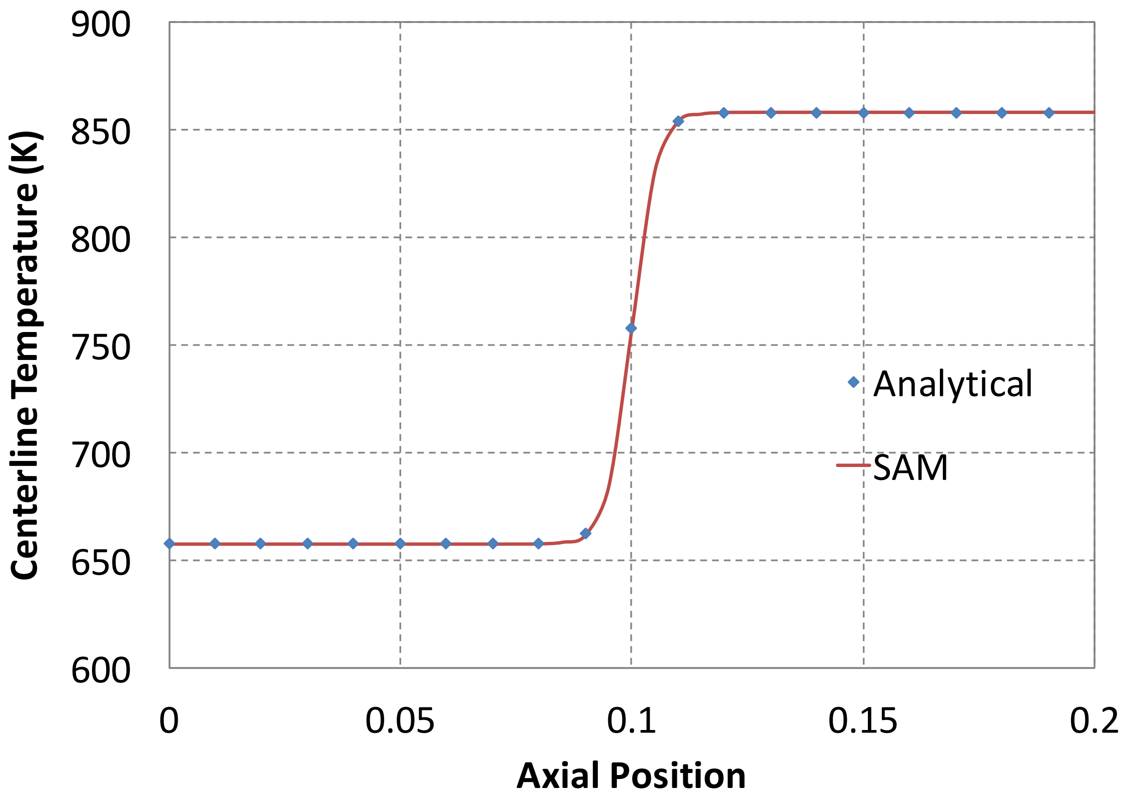


Figure 5.2: Comparisons of centerline temperature distributions of the heated rod, 2D conduction.

The input file of this example problem is shown as follows:

```
[GlobalParams]
  global_init_T = 400
  Tsolid_sf = 1e-1
  [./PBModelParams]
    p_order = 2
  [../]
[]

[Functions]
  [./T_amb_fn]
        type = PiecewiseLinear
        x = '0    0.099    0.101    0.2'
        y = '300    300      500    500'
        axis = x
  [../]
[]

[MaterialProperties]
  [./fuel-mat]
    type = SolidMaterialProps
    k = 2
    Cp = 100
    rho = 1.0e3
  [../]
[]
```

```
[Components]
  [./hs1]
    type = PBCoupledHeatStructure
    position = '0 0 0'
    orientation = '0 0 1'
    hs_type = cylinder

    length = 0.2
    radius_i = 0.0
    width_of_hs = 0.005
    elem_number_radial = 2
    elem_number_axial = 20
    dim_hs = 2
    material_hs = 'fuel-mat'
          heat_source_solid = 6.3661977e7

    Ts_init = 400
    HS_BC_type = 'Adiabatic Convective'
    T_amb_right = T_amb_fn
    Hw_right = 1e3
  [../]
[]

[Postprocessors]
  [./max_T]
    type = NodalMaxValue
    block = 'hs1:hs'
    variable = T_solid
  [../]
[]

[VectorPostprocessors]
  [./Tsolid]
    type = NodalValueSampler
    block = 'hs1:hs'
    variable = T_solid
    sort_by = y
  [../]
[]

[Preconditioning]
   active = 'SMP_PJFNK'
  [./SMP_PJFNK]
    type = SMP
    full = true
    solve_type = 'PJFNK'
  [../]
[] # End preconditioning block

[Executioner]
  type = Steady

  petsc_options_iname = '-ksp_gmres_restart -pc_type'
  petsc_options_value = '100 lu'

  nl_rel_tol = 1e-9
  nl_abs_tol = 1e-6
  nl_max_its = 10

  l_tol = 1e-5 # Relative linear tolerance for each Krylov solve
  l_max_its = 100 # Number of linear iterations for each Krylov solve

   [./Quadrature]
      type = SIMPSON
      order = SECOND
   [../]
[] # close Executioner section

[Outputs]
  perf_graph = true
  print_linear_residuals = false
  [./out_displaced]
    type = Exodus
    use_displaced = true
    execute_on = 'initial timestep_end'
    sequence = false
  [../]
  [./csv]
    type = CSV
  [../]
  [./console]
    type = Console
  [../]
[]
```

5.2 Single Channel Flow

A simple pipe flow problem is presented here, with fixed constant or time-varying boundary conditions. The inlet temperature of the one-meter pipe is fixed at 628 K, or oscillates following a sinusoidal distribution, $T_{in}(t) = 628 + 100\sin(\pi t)$; the inlet velocity is fixed, $u_{in}(t)$=0.5 m/s; and the initial pipe temperate is at 628 K. After executing the test problems, the results can be imported into Paraview for visualization, as shown in Figure 5.3. The transient responses of the inlet temperature wave propagation problem are shown in Figure 5.4, where the code predictions agreed very well with the analytical solutions. This is because of the high-order accuracy in both spatial and temporal (BDF2) discretizations used in SAM.

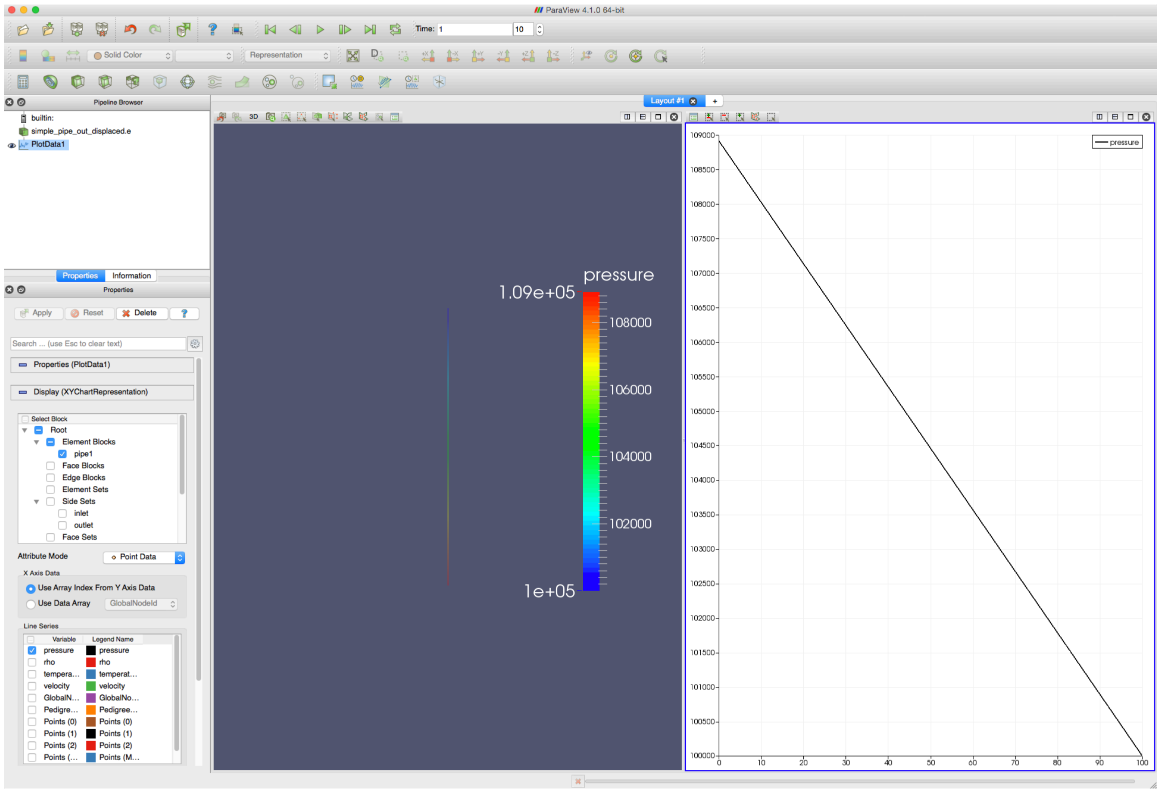


Figure 5.3: Example of SAM results shown in Paraview.

The input file of this example problem is shown as follows:

```
[GlobalParams]
    global_init_P = 1.0e5              # Global initial fluid pressure
    global_init_V = 0.5                # Global initial temperature for fluid and solid
    global_init_T = 628.15             # Global initial fluid velocity
    scaling_factor_var = '1 1e-3 1e-6'  # Scaling factors for fluid variables (p, v, T)
[]

[EOS]
  [./eos]                                     # EOS name
    type = PBSodiumEquationOfState            # Using the sodium equation-of-state
  [../]
[]
```

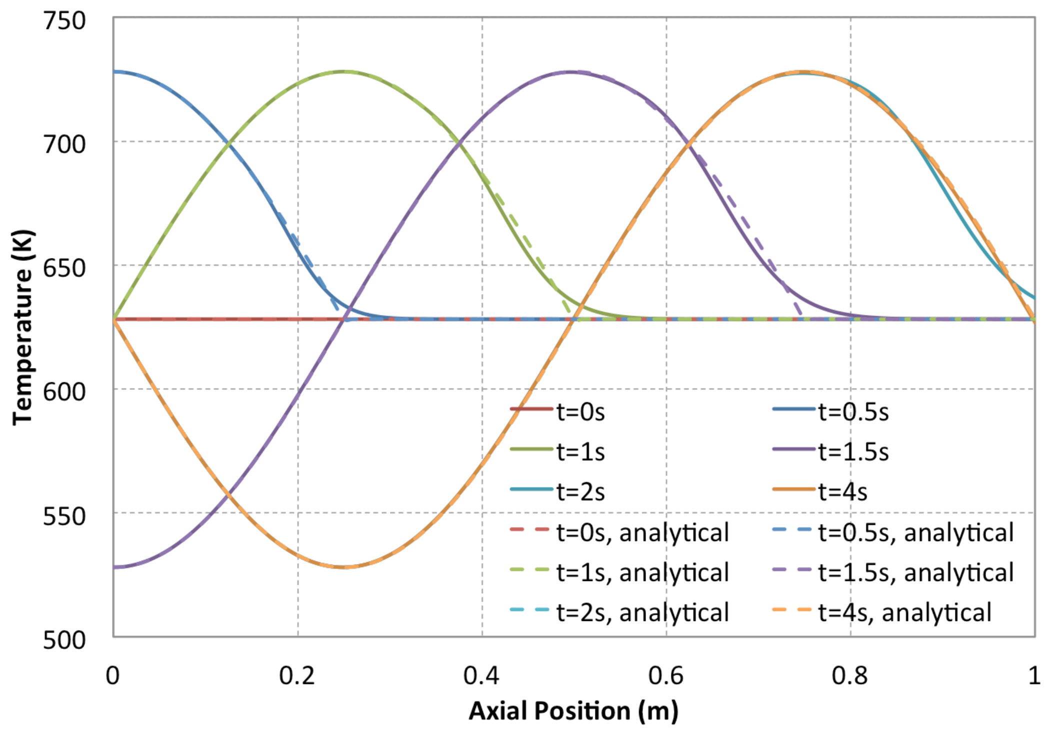


Figure 5.4: Transient responses of the pipe under inlet temperature oscillation, BDF2.

```
[Functions]
  [./tin_sine]                                    # Function name
    type = ParsedFunction                         # Parsed function
    value = 628+100*sin(pi*t)                     # Parsed function formula
  [../]
[]

[Components]
  [./pipe1]
    type = PBOneDFluidComponent
    eos = eos                           # The equation-of-state name
    position = '0 0 0'                  # The origin position of this component
    orientation = '0 0 1'               # The orientation of the component
    heat_source = 0                     # Volumetric heat source
    f=0.01                              # Specified friction coefficient
    Dh = 0.02                           # Equivalent hydraulic diameter
    length = 1                          # Length of the component
    n_elems = 100                       # Number of elements used in discretization
    A = 3.14e-4                         # Area of the One-D fluid component
[../]

  [./inlet]
    type = PBTDJ
    input = 'pipe1(in)'                 # Name of the connected components and the end type
    eos = eos                           # The equation-of-state
    v_bc = 0.5                          # Velocity boundary condition
    T_bc = 628 # or T_fn = tin_sine     # Temperature boundary condition
  [../]

  [./outlet]
```

```
    type = PressureOutlet
    input = 'pipe1(out) '              # Name of the connected components and the end type
    eos = eos                          # The equation-of-state
    p_bc = '1.0e5'                     # Pressure boundary condition
  [../]
[]

[Preconditioning]
   active = 'SMP_PJFNK'
  [./SMP_PJFNK]
    type = SMP                         # Single-Matrix Preconditioner
    full = true                        # Using the full set of couplings among all variables
    solve_type = 'PJFNK'               # Using Preconditioned JFNK solution method
    petsc_options_iname = '-pc_type'   # PETSc option, using preconditiong
    petsc_options_value = 'lu'         # PETSc option, using 'LU' precondition type
                                       # in Krylov solve
  [../]
[] # End preconditioning block

[Executioner]
  type = Transient                  # This is a transient simulation

  dt = 0.02                         # Targeted time step size
  dtmin = 1e-5                      # The allowed minimum time step size

  petsc_options_iname = '-ksp_gmres_restart'  # Additional PETSc settings, name list
  petsc_options_value = '100'                 # Additional PETSc settings, value list

  nl_rel_tol = 1e-7                 # Relative nonlinear tolerance for each Newton solve
  nl_abs_tol = 1e-6                 # Relative nonlinear tolerance for each Newton solve
  nl_max_its = 20                   # Number of nonlinear iterations for each Newton solve

  l_tol = 1e-4                      # Relative linear tolerance for each Krylov solve
  l_max_its = 100                   # Number of linear iterations for each Krylov solve

  start_time = 0.0                  # Physical time at the beginning of the simulation
  num_steps = 200                   # Max. simulation time steps
  end_time = 100.                   # Max. physical time at the end of the simulation

  [./Quadrature]
    type = TRAP                     # Using trapezoid integration rule
    order = FIRST                   # Order of the quadrature
  [../]
[] # close Executioner section

[Outputs]
  [./console]
    type = Console                       # Screen output
    perf_log = true                      # Output the performance log
  [../]
  [./out_displaced]
    type = Exodus                        # Output simulation data to an ExodusII file
    use_displaced = true                 # Use displaced mesh
    execute_on = 'initial timestep_end'  # Output data at the beginning of the simulation
                                         # and each time step
    sequence = false                     # Don't save sequential file output per time step
  [../]
[]
```

5.3 Core Channel

A simple core channel problem (coolant flow and solid conduction in fuel assembly) is presented here with uniform power distribution inside the fuel pin. The schematic of the spatial discretization of the core channel problem is shown in Figure 5.5. The different lines of colors on the left represent different heat structures in an SFR fuel pin (i.e., fuel, sodium gap, and clad). The inlet of the core channel flow is fixed at constant temperature and flow rate. Constant material thermo-physical properties are assumed for this test. Therefore, the analytical solutions of this test problem can be easily derived, with coolant temperature:

$$T_{coolant}(z) = T_{in} + \frac{q'}{\dot{m}c_p} z \tag{5.1}$$

and the fuel centerline temperature:

$$T_{f,cl}(z) = T_{in} + q' \left[\frac{z}{\dot{m}c_p} + \frac{1}{2\pi R_{co} h_c} + \frac{1}{2\pi k_c} \ln\left(\frac{R_{co}}{R_{ci}}\right) + \frac{1}{2\pi R_f h_g} + \frac{1}{4\pi k_f} \right] \tag{5.2}$$

The simulation results can be compared with the analytical solutions as an verification study.

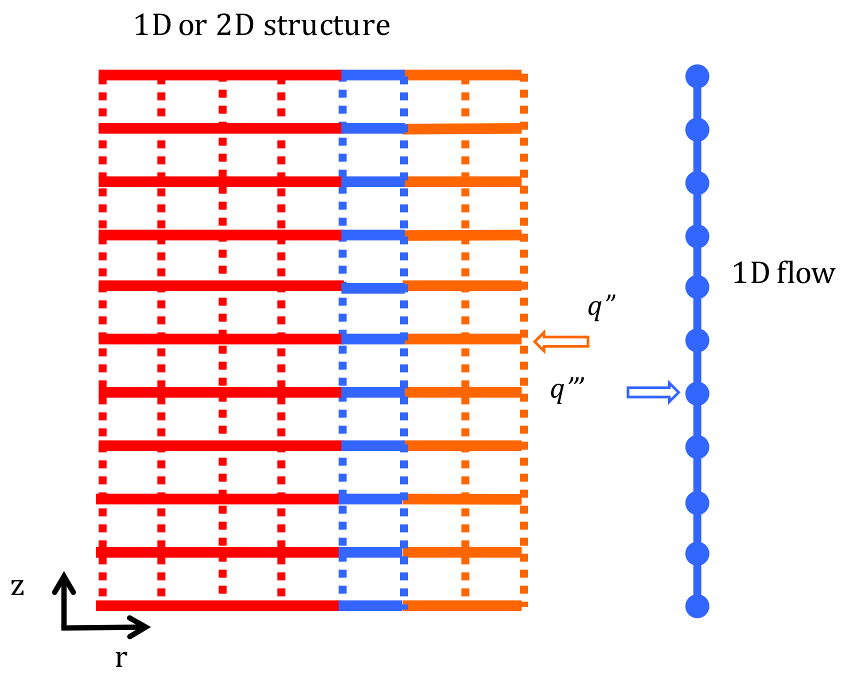


Figure 5.5: The schematic of the spatial discretization of the core channel problem.

The input file of this example problem is shown as follows:

```
[GlobalParams]
  global_init_P = 1.0e5
  global_init_V = 5
  global_init_T = 628.15
  scaling_factor_var = '1 1e-3 1e-6'
  Tsolid_sf = 1e-3                              # Scaling factors for solid temperature
[]

[EOS]
  [./eos]                                       # EOS name
    type = PTConstantEOS
    p_0 = 1e5                                   # Pa, reference pressure
    rho_0 = 865.51                              # kg/m^3, reference density
    beta = 2.7524e-4                            # K^{-1}, thermal expansion coefficient
    cp =  1272.0                                # specific heat;
    h_0 = 7.9898e5                              # J/kg, enthalpy at reference temperature
    T_0 = 628.15                                # K, reference temperature
    mu = 2.6216e-4                              # Pa-s, dynamic viscosity
    k =  72                                     # W/K/m, thermal conductivity
  [../]
[]

[Materials]
  [./fuel-mat]                               # Material name
    type = SolidMaterialProps
    k = 16                                   # Thermal conductivity
    Cp = 191.67                              # Specific heat
    rho = 1.4583e4                           # Density
  [../]
  [./gap-mat]                                # Material name
    type = SolidMaterialProps
    k = 64                                   # Thermal conductivity
    Cp = 1272                                # Specific heat
    rho = 865                                # Density
  [../]
  [./clad-mat]                               # Material name
    type = SolidMaterialProps
    k = 26                                   # Thermal conductivity
    Cp = 638                                 # Specific heat
    rho = 7.646e3                            # Density
  [../]
  [./duct-mat]                               # Material name
    type = SolidMaterialProps
    k = 26                                   # Thermal conductivity
    Cp = 638                                 # Specific heat
    rho = 6e3                                # Density
  [../]
[]

[Functions]
  active = 'uniform'
  [./uniform]                                # Function name
    type = PiecewiseLinear                   # Function type
    axis = 0                                 # X-co-ordinate is used for x
    x = '0  0.8'                             # The x abscissa values
    y = '1  1'                               # The y abscissa values
  [../]
[]

[Components]
  [./reactor]
    type = ReactorPower
    initial_power = 3e4                      # Initial total reactor power
  [../]
  [./CH1]                                    # Component name
    type = PBCoreChannel                     # PBCorechannel component
    eos = eos                                # The equation-of-state name
    position = '0 0 0'
    orientation = '0 0 1'
    A = 2e-05
    Dh = 3.1830989e-3
    length = 0.8
    n_elems = 16

    f = 0.017                              # User specified friction coefficient
    Hw = 1.6e5                             # User specified heat transfer coefficient
    HT_surface_area_density = 1256.637     # Heat transfer surface area density, Ph/Ac

    name_of_hs = 'fuel gap clad'           # Heat structure names
    Ts_init = 628.15                       # Initial structure temperature
    n_heatstruct = 3                       # Number of heat structures
    fuel_type = cylinder                   # Fuel geometric type, cylinder or plate
    width_of_hs = '0.003015 0.000465  0.00052' # The width of all heat structures
    elem_number_of_hs = '20 2 2'           # The element numbers of all heat structures
    material_hs = 'fuel-mat gap-mat clad-mat'  # The material used for all heat structures
    power_fraction = '1.0 0.0 0.0'         # The power fractions of all heat structures
    power_shape_function = uniform            # the axial power shape function name
  [../]

  #Boundary components
  [./inlet]
    type = PBTDJ
    input = 'CH1(in)'
    v_bc = 8.6654
    T_bc = 628.15
    eos = eos
  [../]
  [./outlet]
    type = PBTDV
    input = 'CH1(out)'
    p_bc = '2.0e5'
    T_bc = 728.15
    eos = eos
  [../]
[]

[Postprocessors]
  [./max_Tcoolant]          # Output maximum fluid temperature of block CH1:pipe
    type = NodalMaxValue
    block = 'CH1:pipe'
    variable = temperature
  [../]
  [./max_Tw]                # Output maximum wall temperature of block CH1:pipe
    type = NodalMaxValue
    block = 'CH1:pipe'
    variable = Tw
  [../]
  [./max_Tclad]             # Output maximum solid temperature of block CH1:solid:clad
    type = NodalMaxValue
    block = 'CH1:solid:clad'
    variable = T_solid
  [../]
  [./max_Tf]                # Output maximum solid temperature of block CH1: solid:fuel
    type = NodalMaxValue
    block = 'CH1:solid:fuel'
    variable = T_solid
  [../]
[]

[Preconditioning]
   active = 'SMP_PJFNK'
  [./SMP_PJFNK]
    type = SMP
    full = true
    solve_type = 'PJFNK'
    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]

[] # End preconditioning block


[Executioner]
  type = Steady

  petsc_options_iname = '-ksp_gmres_restart'
  petsc_options_value = '300'

  nl_rel_tol = 1e-9
  nl_abs_tol = 1e-7
  nl_max_its = 20

  l_tol = 1e-5
  l_max_its = 50

   [./Quadrature]
      type = TRAP
      order = FIRST
   [../]
[] # close Executioner section

[Outputs]
  [./out]
    type = Checkpoint                          # Save snapshots of the simulation data
  [../]
  [./console]
    type = Console
    perf_log = true
  [../]
  [./out_displaced]
    type = Exodus
    use_displaced = true
    execute_on = 'initial timestep_end'
    sequence = false
  [../]
[]
```

5.4 Heat Exchanger

An example of a counter-current heat exchanger problem is presented here. The inlet temperatures are 783 K and 606 K for the primary and secondary pipes. The mass flow rates are also fixed at the inlets of the two sides. Because the flow rates are very close for the two sides, linear temperature distributions are expected for the two sides, as the code predictions shown in Figure 5.6.

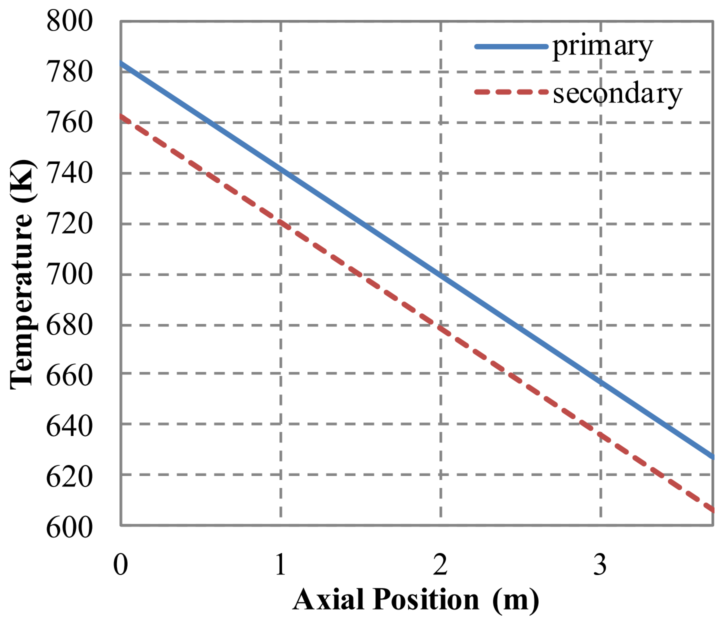


Figure 5.6: Temperature distribution of a counter-current heat exchanger.

The input file of this example problem is shown as follows:

```
[GlobalParams]
  global_init_P = 1.0e5
  global_init_V = 1
  global_init_T = 628.15
  scaling_factor_var = '1 1e-3 1e-6'
  Tsolid_sf = 1e-3
[]

[EOS]
  [./eos]
    type = PBSodiumEquationOfState
  [../]
[]

[Materials]
  [./ss-mat]
    type = SolidMaterialProps
    k = 10
    Cp = 638
    rho = 6e3
  [../]
[]

[Components]
  [./IHX]
```

```
    type = PBHeatExchanger
    eos = eos                                 # EOS of the primary side
    eos_secondary = eos                       # EOS of the secondary side
    position = '0 0 0'
    orientation = '1 0 0'
    A = 0.766                                 # Flow area of the primary side
    A_secondary = 0.517                       # Flow area of the secondary side
    Dh = 0.0186                               # Hydraulic diameter of the primary side
    Dh_secondary = 0.014                      # Hydraulic diameter of the secondary side
    length = 3.71
    n_elems = 20

    Hw = 1.6129e5                             # User specified heat transfer coefficient
                                              # for primary pipe
    Hw_secondary = 1.6129e5                   # User specified heat transfer coefficient
                                              # for secondary pipe
    HTC_geometry_type = Pipe                      # Geometry type of the primary pipe,
                                                  # pipe or bundle
    HTC_geometry_type_secondary = Pipe            # Geometry type of the secondary pipe
    HT_surface_area_density = 729                 # Heat transfer surface area of the
                                                  # primary side, Ph/Ac
    HT_surface_area_density_secondary = 1080.1    # Heat transfer surface area of the
                                                  # secondary side, Ph/Ac

    f = 0.022                       # User specified friction coefficient for primary pipe
    f_secondary = 0.022             # User specified friction coefficient for secondary pipe
    initial_V_secondary = -2        # Initial velocity for secondary pipe

    Twall_init = 628.15             # Initial wall temperature
    wall_thickness = 0.0033         # Wall thickness

    dim_wall = 1                    # Dimensions of the wall, 1D or 2D
    material_wall = ss-mat          # Wall material
    n_wall_elems = 2                # The number of elements in the radial direction
  [../]

  [./inlet1]
    type = PBTDJ
    input = 'IHX(primary_in)'
    eos = eos
    v_bc = 2
    T_bc = 783.15
  [../]

  [./outlet1]
    type = PressureOutlet
    input = 'IHX(primary_out)'
    eos = eos
    p_bc = 1.0e5
  [../]

  [./inlet2]
    type = PBTDJ
    input = 'IHX(secondary_in)'
    eos = eos
    v_bc = -2
    T_bc = 606.15
  [../]

  [./outlet2]
    type = PressureOutlet
    input = 'IHX(secondary_out)'
    eos = eos
    p_bc = 1.0e5
  [../]
[]

[Preconditioning]
   active = 'SMP_PJFNK'

  [./SMP_PJFNK]
    type = SMP
    full = true
    solve_type = 'PJFNK'
    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]

[] # End preconditioning block

[Postprocessors]
  # The total heat removal rate at the primary side of IHX
  [./heat_removal_primary]
    type = HeatExchangerHeatRemovalRate
    block = IHX:primary_pipe
    heated_perimeter = 558.414
  [../]
  # The total heat removal rate at the secondary side of IHX
  [./heat_removal_secondary]
    type = HeatExchangerHeatRemovalRate
    block = IHX:secondary_pipe
    heated_perimeter = 558.414
  [../]
[]

[Executioner]
  type = Transient # try Steady solver as well

  dt = 0.2
  dtmin = 1e-4

  petsc_options_iname = '-ksp_gmres_restart'
  petsc_options_value = '101'

  nl_rel_tol = 1e-8
  nl_abs_tol = 1e-6
  nl_max_its = 30

  l_tol = 1e-4
  l_max_its = 100

  start_time = 0.0
  num_steps = 10
  end_time = 100.

   [./Quadrature]
      type = TRAP
      order = FIRST
   [../]
[] # close Executioner section

[Outputs]
  [./out_displaced]
    type = Exodus
    use_displaced = true
    execute_on = 'initial timestep_end'
    sequence = false
  [../]
  [./console]
    type = Console
    perf_log = true
  [../]
[]
```

5.5 Volume Branch

An example problem with a VolumeBranch component included is presented here. The boundary and the initial conditions of the five pipes are shown in Figure 5.7 and Figure 5.8. Note that very different inlet orifice coefficients have been used for the connecting nodes. The volume of the PBVolumeBranch is 0.0314 m^3, and the initial temperature of the volume is at 628.15 K. Because Pipe 3 has very high inlet flow rate but low inlet temperature, the temperature at the VolumeBranch, outlet of Pipes 4 and 5 will decrease correspondingly, as shown in Figure 5.9.

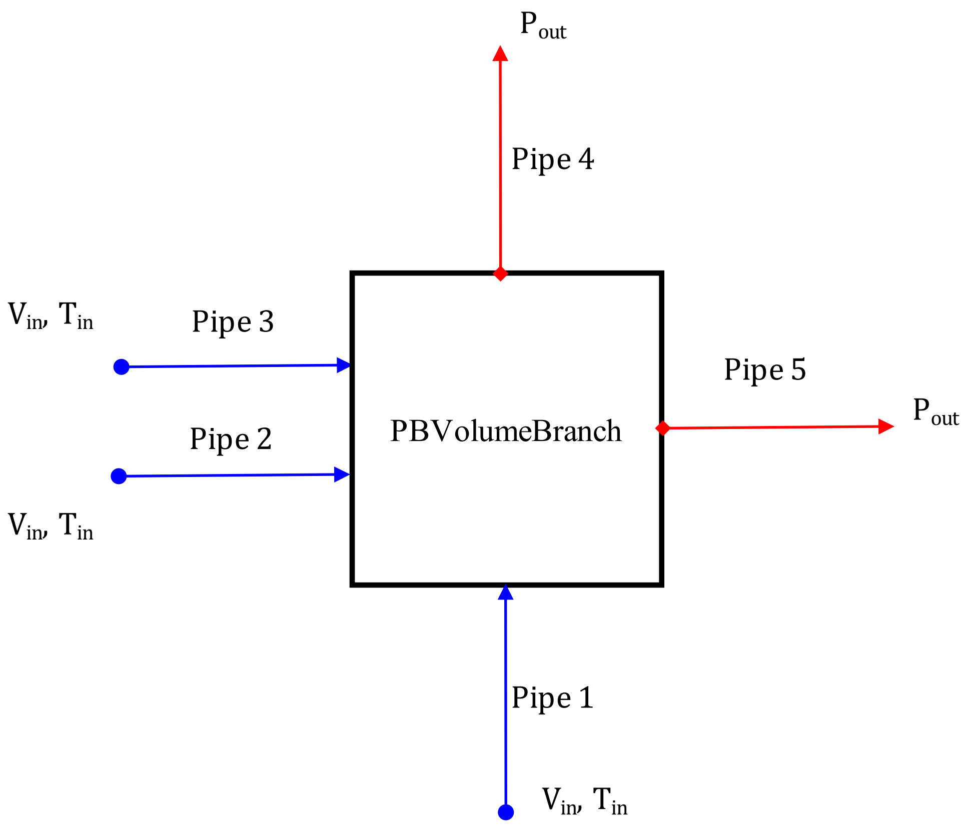


Figure 5.7: The three-pipe-in and two-pipe-out VolumeBranch test model.

Component	Pipe 1	Pipe 2	Pipe 3	Pipe 4	Pipe 5
Length (m)	1	1	1	1	1
Diameter (m)	0.02	0.02	0.02	0.02	0.02
Boundary Type	Flow inlet	Flow inlet	Flow inlet	Pressure outlet	Pressure outlet
Orifice Coeff. to VolumeBranch	0.01	0.01	0.01	0.01	100
Z-coordinate of the node connected to VolumeBranch (m)	0	0.25	0.75	1	0.5
Boundary Conditions	Vin =1 m/s Tin = 628.15K	Vin =1 m/s Tin = 628.15K	Vin =10 m/s Tin = 528.15K	Pout = 10^5 Pa Tout = 628.15K	Pout = 1.5×10^5 Pa Tout = 628.15K
Initial Conditions					
Pressure (Pa)	1.5×10^5				
Velocity (m/s)	1				
Temperature (K)	628.15				

Figure 5.8: Input parameters of the three pipe in and two pipe out VolumeBranch test model.

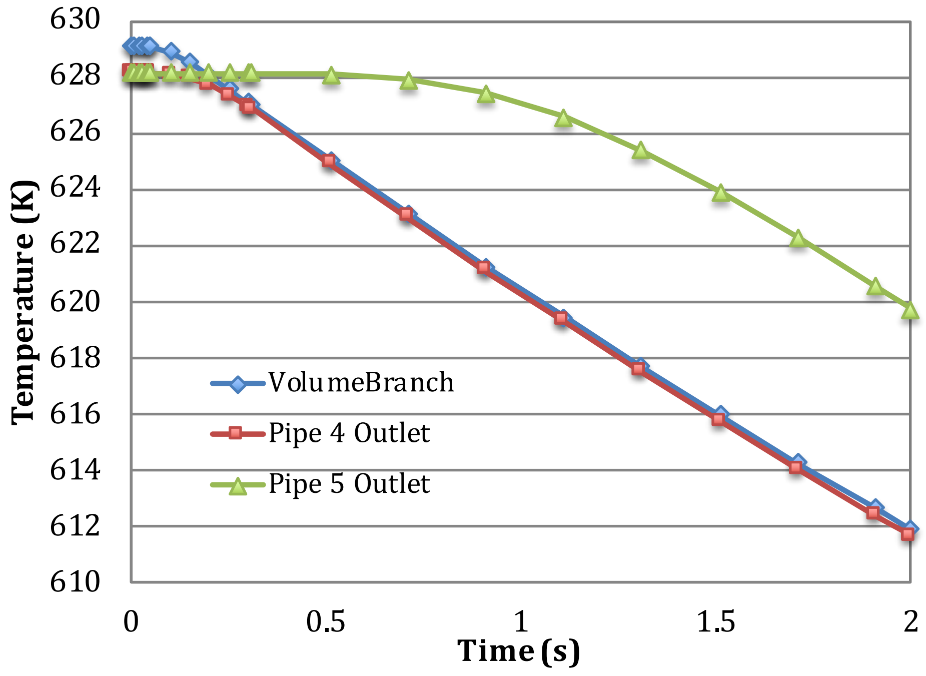


Figure 5.9: Transient temperature response at the VolumeBranch and pipe outlets.

The input file of this example problem is shown as follows:

```
[GlobalParams]
```

```
    global_init_P = 1.2e5
    global_init_V = 1
    global_init_T = 628.15
    scaling_factor_var = '1 1e-3 1e-6'
[]

[EOS]
  active = 'eos'
  [./eos]
    type = PBSodiumEquationOfState
  [../]
[]

[Components]
  [./pipe1]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 0 0'
    orientation = '0 0 1'

    A = 3.14e-4
    Dh = 0.02
    length = 1
    n_elems = 10
    f = 0.01
    Hw = 0
  [../]

  [./pipe2]
    type = PBOneDFluidComponent
    eos = eos
    position = '-1.5 0 1.25'
    orientation = '1 0 0'

    A = 3.14e-4
    Dh = 0.02
    length = 1
    n_elems = 10
    f = 0.01
    Hw = 0
  [../]

  [./pipe3]
    type = PBOneDFluidComponent
    eos = eos
    position = '-1.5 0 1.75'
    orientation = '1 0 0'
    initial_T = 528.15

    A = 3.14e-4
    Dh = 0.02
    length = 1
    n_elems = 10
    f = 0.01
    Hw = 0
  [../]

  [./pipe4]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 0 2'
    orientation = '0 0 1'

    A = 3.14e-4
    Dh = 0.02
    length = 1
    n_elems = 10
    f = 0.01
    Hw = 0
  [../]

  [./pipe5]
    type = PBOneDFluidComponent
    eos = eos
    position = '0.5 0 1.5'
    orientation = '1 0 0'

    A = 3.14e-4
    Dh = 0.02
    length = 1
    n_elems = 10
    f = 0.01
    Hw = 0
  [../]

  [./branch1]
    type = PBVolumeBranch
    eos = eos
    center = '0 0 1.5'     # The center or reference position of the volume branch

    inputs = 'pipe1(out) pipe2(out) pipe3(out)'# The input connections of the volume branch
    outputs = 'pipe4(in) pipe5(in)'          # The output connections of the volume branch
    K = '0.01 0.01 0.01 0.01 100'            # The form loss coefficient at all connections

    Area = 3.14e-2                           # Reference flow area
    volume = 3.14e-2                         # Total volume
    initial_T = 628.15                       # Initial volume temperature
[../]

  [./inlet1]
    type = PBTDJ
    input = 'pipe1(in)'
    eos = eos
    v_bc = 1.0
    T_bc = 628.15
  [../]
  [./inlet2]
    type = PBTDJ
    input = 'pipe2(in)'
    eos = eos
    v_bc = 1.0
    T_bc = 628.15
  [../]
  [./inlet3]
    type = PBTDJ
    input = 'pipe3(in)'
    eos = eos
    v_bc = 10.0
    T_bc = 528.15
  [../]
  [./outlet1]
    type = PBTDV
    input = 'pipe4(out)'
    eos = eos
    p_bc = '1.0e5'
    T_bc = 628.15
  [../]
  [./outlet2]
```

```
    type = PBTDV
    input = 'pipe5(out)'
    eos = eos
    p_bc = '1.5e5'
    T_bc = 628.15
  [../]
[]

[Preconditioning]
  [./SMP_PJFNK]
    type = SMP
    full = true
    solve_type = 'PJFNK'
    petsc_options_iname = '-pc_type '
    petsc_options_value = 'lu'
  [../]
[] # End preconditioning block

[Executioner]
  type = Transient

  dt = 1e-1
  dtmin = 1e-5

  # setting time step range
  # Time step size is controlled by this TimeStepper
  [./TimeStepper]
    type = FunctionDT
    time_t = '    0   0.1  0.2    20    21   100   101    1e5'    # Physical time
    time_dt ='0.01 0.01  0.1   0.1   0.5   0.5     1      1'    # Time step size dependent on
                                                                  # the physical time
  [../]

  petsc_options_iname = '-ksp_gmres_restart'
  petsc_options_value = '100'

  nl_rel_tol = 1e-8
  nl_abs_tol = 1e-7
  nl_max_its = 20

  l_tol = 1e-5
  l_max_its = 100

  start_time = 0.0
  num_steps = 100
  end_time = 2.

   [./Quadrature]
      type = TRAP
      order = FIRST
   [../]
[] # close Executioner section

[Outputs]
  [./out_displaced]
    type = Exodus
    use_displaced = true
    execute_on = 'initial timestep_end'
    sequence = false
  [../]

  [./console]
    type = Console
    perf_log = true
  [../]
[]
```

5.6 A Simple Loop Model

An example problem with a simple loop problem is presented here. It consists of six 1-D pipes (PBOneDFluidComponent) and a heat exchanger (PBHeatExchanger). One pipe is internally heated, as shown in Figure 5.10. The primary loop (including the heat exchanger) is connected by a set of PBSingleJunctions, a PBBranch, and a Pump. The secondary side of the heat exchanger has fixed inlet velocity and temperature and fixed outlet pressure boundary conditions. Note that if the Pump is replaced by a PBBranch, the loop will be derived by natural circulation.

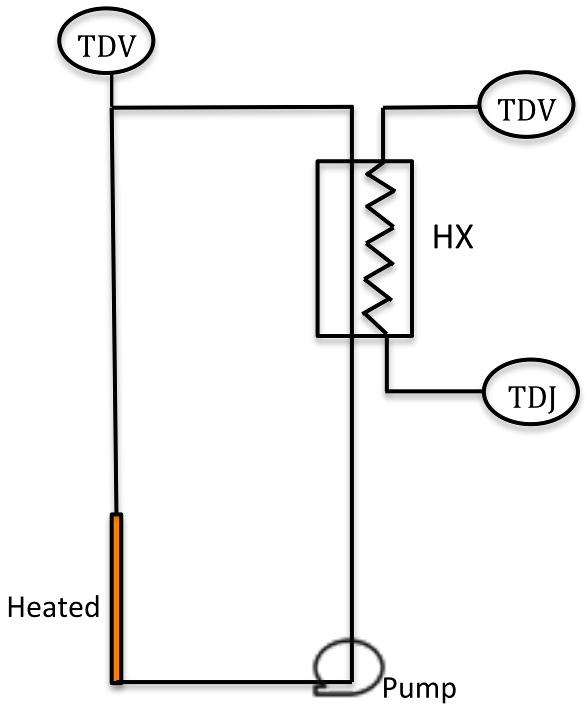


Figure 5.10: Schematics of the a test loop problem.

The input file of this example problem is shown as follows:

```
[GlobalParams]
  global_init_P = 1.1e5
  global_init_V = 0.1
  global_init_T = 628.15
  Tsolid_sf = 1e-1

  [./PBModelParams]
    pbm_scaling_factors = '1 1e-3 1e-6'
  [../]
[]

[EOS]
  [./eos]
    type = PBSodiumEquationOfState
  [../]
[]

[Materials]
  [./ss-mat]
```

```
    type = SolidMaterialProps
    k = 20
    Cp = 638
    rho = 6e3
  [../]
[]

[Components]
  [./pipe1]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 1 0'
    orientation = '0 -1 0'

    A = 0.44934
    Dh = 2.972e-3
    length = 1
    n_elems = 10
    f = 0.001
  [../]

  [./CH1]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 0 0'
    orientation = '0 0 1'

    A = 0.44934
    Dh = 2.972e-3
    length = 0.8
    n_elems = 10

    f = 0.022 #McAdams
    heat_source = 5e7
  [../]

  [./pipe2]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 0 0.8'
    orientation = '0 0 1'

    A = 0.44934
    Dh = 2.972e-3
    length = 5.18
    n_elems = 10
    f = 0.001
  [../]

  [./pipe3]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 0 5.98'
    orientation = '0 1 0'

    A = 0.44934
    Dh = 2.972e-3
    length = 1
    n_elems = 10
    f = 0.001
  [../]

  [./IHX]
    type = PBHeatExchanger
    eos = eos
    eos_secondary = eos

    position = '0 0.976 5.98'
    orientation = '0 0 -1'
    A = 0.44934
    Dh = 0.0186
    A_secondary = 0.44934
    Dh_secondary = 0.0186
    length = 0.8
    n_elems = 20
    f = 0.022

    initial_V_secondary = -0.2

    HT_surface_area_density = 1e3
    HT_surface_area_density_secondary = 1e3

    Twall_init = 628.15
    wall_thickness = 0.004

    dim_wall = 1
    material_wall = ss-mat
    n_wall_elems = 2
  [../]

  [./pipe4]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 1.0 5.18'
    orientation = '0 0 -1'

    A = 0.44934
    Dh = 2.972e-3
    length = 5.18
    n_elems = 10
    f = 0.001
  [../]

  [./Branch1]
    type = PBSingleJunction
    inputs = 'pipe1(out)'
    outputs = 'CH1(in) '
    eos = eos
  [../]
  [./Branch2]
    type = PBSingleJunction
    inputs = 'CH1(out) '
    outputs = 'pipe2(in)'
    eos = eos
  [../]
  [./Branch3]
    type = PBBranch
    inputs = 'pipe2(out)'
    outputs = 'pipe3(in) pipe5(in)'
    K = '0.0 0.0 10.0'
    Area =   0.44934
    initial_P = 1e5
    eos = eos
  [../]
  [./Branch4]
    type = PBSingleJunction
    inputs = 'pipe3(out)'
    outputs = 'IHX(primary_in)'
    eos = eos
  [../]
  [./Branch5]
    type = PBSingleJunction
    inputs = 'IHX(primary_out)'
    outputs = 'pipe4(in)'
    eos = eos
  [../]

###### switch between Brach6 and Pump_p for natural circulation or forced flow

#  [./Pump_p]
#    type = PBPump                          # This is a PBPump component
#    eos = eos
#    inputs = 'pipe4(out)'
#    outputs = 'pipe1(in)'
#    K = '1. 1.'                            # Form loss coefficient at pump inlet and outlet
#    Area = 0.44934                         # Reference pump flow area
#    initial_P = 1.5e5                      # Initial pressure
#    Head = 5e3                             # Pump head, Pa
#  [../]

  [./Branch6]
    type = PBSingleJunction
    inputs = 'pipe4(out)'
    outputs = 'pipe1(in)'
    eos = eos
  [../]

  [./pipe5]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 0 5.98'
    orientation = '0 0 1'
    A = 0.44934
    Dh = 2.972e-3
    length = 0.1
    n_elems = 2
    f = 0.001
  [../]
  [./p_out]
    type = PressureOutlet
    input = 'pipe5(out)'
    eos = eos
    p_bc = '1e5'
  [../]

  [./inlet2]
    type = PBTDJ
    input = 'IHX(secondary_in)'
    eos = eos
    v_bc = -1
    T_bc = 606.15
  [../]

  [./outlet2]
    type = PressureOutlet
    input = 'IHX(secondary_out)'
    eos = eos
    p_bc = 1.0e5
  [../]
[]

[Postprocessors]
```

```
  # Output mass flow rate at inlet of CH1
  [./CH1_flow]
    type = ComponentBoundaryFlow
    input = CH1(in)
  [../]
[]

[Preconditioning]
  active = 'SMP_PJFNK'
  [./SMP_PJFNK]
    type = SMP
    full = true
    solve_type = 'PJFNK'
    petsc_options_iname = '-pc_type -ksp_gmres_restart'
    petsc_options_value = 'lu 101'
  [../]
[]

[Executioner]
  type = Steady

  nl_rel_tol = 1e-8
  nl_abs_tol = 1e-7
  nl_max_its = 20
  l_tol = 1e-6
  l_max_its = 100

  [./Quadrature]
    type = TRAP
    order = FIRST
  [../]
[]

[Outputs]
  print_linear_residuals = false
  [./out_displaced]
    type = Exodus
    use_displaced = true
    execute_on = 'initial timestep_end'
    sequence = false
  [../]

  [./console]
    type = Console
    perf_log = true
  [../]
[]
```

5.7 A Simplified SFR Model

A typical pool-type SFR test problem is presented here, based on the design information of the Advanced Burner Test Reactor (ABTR) conceptual design [18]. Figure 5.11 shows the schematics of the test SFR model. The primary coolant system consists of the downcomers (pump outlet and pump discharge), the lower plenum, the reactor core model, the upper plenum, and the intermediate heat exchanger. Five PBCoreChannels are used to describe the reactor core. PBLiquidVolume components are used to represent the cold pool and the upper plenum. Both are connected to a CoverGas component. Different components are connected with junction Components. The intermediate loop, the secondary loop, and the DRACS loop are modeled with great simplicities. Single-phase counter current heat exchanger models (PBHeatExchanger) are used to mimic the function of the intermediate loop heat exchanger (IHX), DRACS heat exchanger (DHX), and secondary loop heat exchanger (SHX) to transfer heat among the primary, intermediate, secondary, and the DRACS loops.

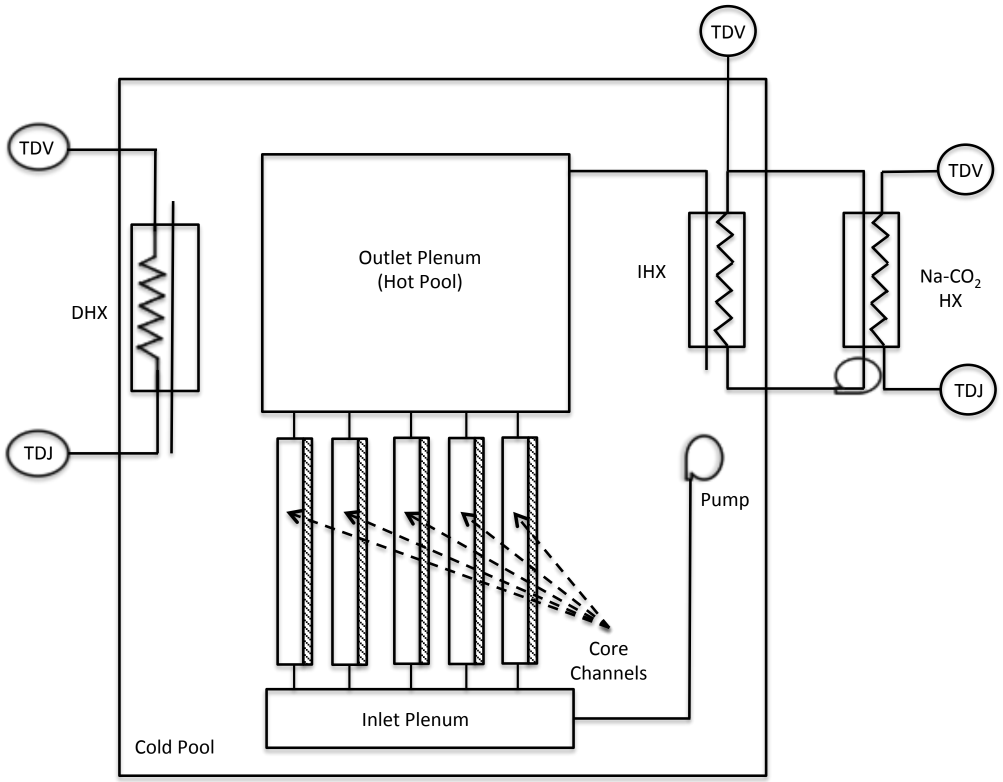


Figure 5.11: Schematics of the a simple pool-type SFR model.

The input file of this example problem is shown as follows:

```
[GlobalParams]
    global_init_P = 2e5
    global_init_V = 0.1
    global_init_T = 628.15
    Tsolid_sf = 1e-3

  [./PBModelParams]
    pbm_scaling_factors = '1 1e-3 1e-6'
    pspg = false
    p_order = 2
  [../]
[]

[EOS]
  [./eos]
    type = PBSodiumEquationOfState
  [../]
[]

[Functions]
  [./ppf_axial]
    type = PiecewiseLinear
    x = '0.0  0.0200  0.0600  0.100  0.140  0.180  0.220  0.260  0.300  0.340    0.380
      0.420  0.460  0.500  0.540  0.580  0.620  0.660  0.700  0.740    0.780   0.800'
    y = '7.818e-1 8.12035e-1 8.72501e-1 9.43054e-1 1.01107 1.04739 1.09779 1.13790
     1.16662 1.17569 1.18022
       1.17255 1.15267 1.13305 1.08829 1.03142 9.62681e-1 9.08601e-1 8.11380e-1
     7.04156e-1 5.90929e-1 5.34316e-1'
    axis = 0
  [../]

  [./power_history]
    type = PiecewiseLinear
  x ='-1.000E+03  5.000E-01  1.000E+00  1.500E+00  2.000E+00  2.500E+00  3.000E+00
    3.500E+00  4.000E+00  4.500E+00
  5.000E+00  5.500E+00  6.000E+00  6.500E+00  7.000E+00  7.500E+00  8.000E+00  8.500E+00
    9.000E+00  9.500E+00
  1.000E+01  1.050E+01  1.100E+01  1.150E+01  1.200E+01  1.250E+01  1.300E+01  1.350E+01
    1.400E+01  1.450E+01
  1.500E+01  1.550E+01  1.600E+01  1.650E+01  1.700E+01  1.750E+01  1.800E+01  1.850E+01
    1.900E+01  1.950E+01
  2.000E+01  2.500E+01  3.000E+01  3.500E+01  4.000E+01  4.500E+01  5.000E+01  5.500E+01
    6.000E+01  6.500E+01
  7.000E+01  7.500E+01  8.000E+01  8.500E+01  9.000E+01  9.500E+01  1.000E+02  1.100E+02
    1.200E+02  1.300E+02
  1.400E+02  1.500E+02  1.600E+02  1.700E+02  1.800E+02  1.900E+02  2.000E+02  2.100E+02
    2.200E+02  2.300E+02
  2.400E+02  2.500E+02  2.600E+02  2.700E+02  2.800E+02  2.900E+02  3.000E+02  3.200E+02
    3.400E+02  3.600E+02
  3.800E+02  4.000E+02  4.500E+02  5.000E+02  5.500E+02  6.000E+02  6.500E+02  7.500E+02
    1.000E+03  2.000E+03
  4.000E+03  6.000E+03  8.000E+03  1.000E+04  1.500E+04  2.000E+04  2.500E+04  3.000E+04
    3.500E+04  4.000E+04 1e5'

  y ='1.000E+00  1.000E+00  9.969E-01  9.892E-01  9.784E-01  1.537E-01  1.390E-01
    1.292E-01  1.215E-01  1.152E-01
  1.097E-01  1.050E-01  1.008E-01  9.710E-02  9.377E-02  9.077E-02  8.807E-02  8.561E-02
    8.337E-02  8.132E-02
  7.944E-02  7.771E-02  7.611E-02  7.463E-02  7.325E-02  7.197E-02  7.077E-02  6.964E-02
    6.858E-02  6.758E-02
  6.663E-02  6.574E-02  6.489E-02  6.408E-02  6.331E-02  6.258E-02  6.187E-02  6.120E-02
    6.055E-02  5.993E-02
  5.933E-02  5.431E-02  5.049E-02  4.741E-02  4.485E-02  4.267E-02  4.078E-02  3.914E-02
    3.769E-02  3.640E-02
  3.526E-02  3.425E-02  3.334E-02  3.252E-02  3.178E-02  3.111E-02  3.051E-02  2.946E-02
    2.858E-02  2.784E-02
  2.720E-02  2.665E-02  2.617E-02  2.575E-02  2.537E-02  2.502E-02  2.471E-02  2.443E-02
    2.417E-02  2.393E-02
  2.370E-02  2.349E-02  2.329E-02  2.310E-02  2.292E-02  2.276E-02  2.259E-02  2.229E-02
    2.201E-02  2.175E-02
  2.151E-02  2.128E-02  2.076E-02  2.029E-02  1.987E-02  1.948E-02  1.912E-02  1.847E-02
    1.715E-02  1.395E-02
```

```
1.112E-02  9.789E-03  8.994E-03  8.448E-03  7.579E-03  7.040E-03  6.657E-03  6.359E-03
  6.117E-03  5.911E-03 4e-3'
[../]

[./pump_p_coastdown]
  type = PiecewiseLinear
x ='-1.000E+03  0.00E+00  4.00E-01  8.00E-01  1.20E+00  1.60E+00  2.00E+00  2.40E+00
  2.80E+00  3.20E+00  3.60E+00
4.00E+00  4.40E+00  4.80E+00  5.20E+00  5.60E+00  6.00E+00  6.40E+00  6.80E+00
  7.20E+00  7.60E+00
8.000E+00  1.000E+01  2.000E+01  3.000E+01  4.000E+01  5.000E+01  6.000E+01  7.000E+01
  8.000E+01  9.000E+01
1.000E+02  1.100E+02  1.200E+02  1.300E+02  1.400E+02  1.500E+02  1.600E+02  1.700E+02
  1.800E+02  1.900E+02
2.000E+02  2.100E+02  2.200E+02  2.300E+02  2.400E+02  2.500E+02  2.600E+02  2.700E+02
  2.800E+02  2.900E+02
3.000E+02  3.100E+02  3.200E+02  3.300E+02  3.400E+02  3.500E+02  3.600E+02  3.700E+02
  3.800E+02  3.900E+02
4.000E+02  4.100E+02  4.200E+02  1.00E+05'

y ='1.000E+00  1.000E+00  9.671E-01  9.355E-01  9.050E-01  8.757E-01  8.476E-01
  8.205E-01  7.945E-01  7.695E-01  7.455E-01
7.225E-01  7.004E-01  6.792E-01  6.590E-01  6.395E-01  6.209E-01  6.031E-01  5.860E-01
  5.697E-01  5.540E-01
5.396E-01  4.749E-01  2.753E-01  1.773E-01  1.219E-01  8.812E-02  6.655E-02  5.206E-02
  4.181E-02  3.425E-02
2.850E-02  2.401E-02  2.043E-02  1.754E-02  1.516E-02  1.317E-02  1.151E-02  1.009E-02
  8.869E-03  7.816E-03
6.898E-03  6.094E-03  5.382E-03  4.752E-03  4.192E-03  3.692E-03  3.253E-03  2.814E-03
  2.480E-03  2.132E-03
1.866E-03  1.621E-03  1.397E-03  1.190E-03  9.999E-04  8.248E-04  6.642E-04  5.175E-04
  3.841E-04  2.637E-04
1.558E-04  5.989E-05    0      0'

scale_factor = 415100
[../]

[./pump_s_coastdown]
  type = PiecewiseLinear
x ='-1.000E+03  0.00E+00  4.00E-01  8.00E-01  1.20E+00  1.60E+00  2.00E+00  2.40E+00
  2.80E+00  3.20E+00  3.60E+00
4.00E+00  4.40E+00  4.80E+00  5.20E+00  5.60E+00  6.00E+00  6.40E+00  6.80E+00
  7.20E+00  7.60E+00
8.000E+00  1.000E+01  2.000E+01  3.000E+01  4.000E+01  5.000E+01  6.000E+01  7.000E+01
  8.000E+01  9.000E+01
1.000E+02  1.100E+02  1.200E+02  1.300E+02  1.400E+02  1.500E+02  1.600E+02  1.700E+02
  1.800E+02  1.900E+02
2.000E+02  2.100E+02  2.200E+02  2.300E+02  2.400E+02  2.500E+02  2.600E+02  2.700E+02
  2.800E+02  2.900E+02
3.000E+02  3.100E+02  3.200E+02  3.300E+02  3.400E+02  3.500E+02  3.600E+02  3.700E+02
  3.800E+02  3.900E+02
4.000E+02  4.100E+02  4.200E+02  1.00E+05'

y ='1.000E+00  1.000E+00  9.671E-01  9.355E-01  9.050E-01  8.757E-01  8.476E-01
  8.205E-01  7.945E-01  7.695E-01  7.455E-01
7.225E-01  7.004E-01  6.792E-01  6.590E-01  6.395E-01  6.209E-01  6.031E-01  5.860E-01
  5.697E-01  5.540E-01
5.396E-01  4.749E-01  2.753E-01  1.773E-01  1.219E-01  8.812E-02  6.655E-02  5.206E-02
  4.181E-02  3.425E-02
2.850E-02  2.401E-02  2.043E-02  1.754E-02  1.516E-02  1.317E-02  1.151E-02  1.009E-02
  8.869E-03  7.816E-03
6.898E-03  6.094E-03  5.382E-03  4.752E-03  4.192E-03  3.692E-03  3.253E-03  2.814E-03
  2.480E-03  2.132E-03
1.866E-03  1.621E-03  1.397E-03  1.190E-03  9.999E-04  8.248E-04  6.642E-04  5.175E-04
```

```
    3.841E-04  2.637E-04
  1.558E-04  5.989E-05    0      0'

  scale_factor = 40300
  [../]

  [./flow_secondary]
    type = PiecewiseLinear
    x ='-1.000E+03  0          1  1e5'
    y = '-1259     -1259      0      0'
    scale_factor = 0.002216 # 1/rhoA
  [../]

  [./flow_dhx]
    type = PiecewiseLinear
    x ='-1.000E+03   0     1      1e5'
    y = '0  0  -6.478    -6.478'
    scale_factor = 0.046 # 1/rhoA
  [../]
[]

[Materials]
  [./fuel-mat]
    type = HeatConductionMaterialProps
    k = 29.3
    Cp = 191.67
    rho = 1.4583e4
  [../]
  [./gap-mat]
    type = HeatConductionMaterialProps
    k = 64
    Cp = 1272
    rho = 865
  [../]
  [./clad-mat]
    type = HeatConductionMaterialProps
    k = 26.3
    Cp = 638
    rho = 7.646e3
  [../]
  [./ss-mat]
    type = HeatConductionMaterialProps
    k = 26.3
    Cp = 638
    rho = 7.646e3
  [../]
[]

[Components]
  [./reactor]
    type = ReactorPower
    initial_power = 250e6
    decay_heat = power_history
  [../]

######  Primary Loop  ######

  [./CH1]
    type = PBCoreChannel
    eos = eos
    position = '0 -1 0'
    orientation = '0 0 1'

    A = 4.9237e-3
    Dh = 2.972e-3
    length = 0.8
    n_elems = 4

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density = 1107.8

    dim_hs = 1
    name_of_hs = 'fuel gap clad'
    Ts_init = 628.15
    n_heatstruct = 3
    fuel_type = cylinder
    width_of_hs = '0.003015 0.000465  0.00052'
    elem_number_of_hs = '2 1 1'
    material_hs = 'fuel-mat gap-mat clad-mat'

    power_fraction = '0.02248 0.0 0.0'
    power_shape_function = ppf_axial
  [../]

  [./CH1_LP]
    type = PBPipe
    eos = eos
    position = '0 -1 -0.6'
    orientation = '0 0 1'

    A = 4.9237e-3
    Dh = 2.972e-3
    length = 0.6
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005 #0.002
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]

  [./CH1_UP]
    type = PBPipe
    eos = eos
    position = '0 -1 0.8'
    orientation = '0 0 1'

    A = 4.9237e-3
    Dh = 2.972e-3
    length = 1.5
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./Branch_CH1_L]
    type = PBSingleJunction
    inputs = 'CH1_LP(out)'
    outputs = 'CH1(in)'
    eos = eos
  [../]
  [./Branch_CH1_U]
    type = PBSingleJunction
    inputs = 'CH1(out)'
    outputs = 'CH1_UP(in)'
    eos = eos
  [../]

  [./CH2]
    type = PBCoreChannel
    eos = eos
    position = '0 -0.5 0'
    orientation = '0 0 1'

    A = 0.11323
    Dh = 2.972e-3
    length = 0.8
    n_elems = 4

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density = 1107.8

    dim_hs = 1
    name_of_hs = 'fuel gap clad'
    Ts_init = 628.15
    n_heatstruct = 3
    fuel_type = cylinder
    width_of_hs = '0.003015 0.000465  0.00052'
    elem_number_of_hs = '2 1 1'
    material_hs = 'fuel-mat gap-mat clad-mat'

    power_fraction = '0.41924 0.0 0.0'
    power_shape_function = ppf_axial
  [../]
  [./CH2_LP]
    type = PBPipe
    eos = eos
    position = '0 -0.5 -0.6'
    orientation = '0 0 1'

    A = 0.11323
    Dh = 2.972e-3
    length = 0.6
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model
    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./CH2_UP]
    type = PBPipe
    eos = eos
    position = '0 -0.5 0.8'
    orientation = '0 0 1'

    A = 0.11323
    Dh = 2.972e-3
    length = 1.5
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./Branch_CH2_L]
    type = PBSingleJunction
    inputs = 'CH2_LP(out)'
    outputs = 'CH2(in)'
    eos = eos
  [../]
  [./Branch_CH2_U]
    type = PBSingleJunction
    inputs = 'CH2(out)'
    outputs = 'CH2_UP(in)'
    eos = eos
  [../]

  [./CH3]
    type = PBCoreChannel
    eos = eos
    position = '0 0 0'
    orientation = '0 0 1'

    A = 0.029539
    Dh = 2.972e-3
    length = 0.8
    n_elems = 4

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density = 1107.8

    dim_hs = 1
    name_of_hs = 'fuel gap clad'
    Ts_init = 628.15
    n_heatstruct = 3
    fuel_type = cylinder
    width_of_hs = '0.003015 0.000465  0.00052'
    elem_number_of_hs = '2 1 1'
    material_hs = 'fuel-mat gap-mat clad-mat'

    power_fraction = '0.09852 0.0 0.0'
    power_shape_function = ppf_axial
  [../]
  [./CH3_LP]
    type = PBPipe
    eos = eos
    position = '0 0 -0.6'
    orientation = '0 0 1'

    A = 0.029539
    Dh = 2.972e-3
    length = 0.6
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./CH3_UP]
    type = PBPipe
    eos = eos
    position = '0 0 0.8'
    orientation = '0 0 1'

    A = 0.029539
    Dh = 2.972e-3
    length = 1.5
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./Branch_CH3_L]
    type = PBSingleJunction
    inputs = 'CH3_LP(out)'
    outputs = 'CH3(in)'
    eos = eos
  [../]
  [./Branch_CH3_U]
```

```
    type = PBSingleJunction
    inputs = 'CH3(out)'
    outputs = 'CH3_UP(in)'
    eos = eos
  [../]


  [./CH4]
    type = PBCoreChannel
    eos = eos
    position = '0 0.5 0'
    orientation = '0 0 1'

    A = 0.14769
    Dh = 2.972e-3
    length = 0.8
    n_elems = 4

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density = 1107.8

    dim_hs = 1
    name_of_hs = 'fuel gap clad'
    Ts_init = 628.15
    n_heatstruct = 3
    fuel_type = cylinder
    width_of_hs = '0.003015 0.000465  0.00052'
    elem_number_of_hs = '2 1 1'
    material_hs = 'fuel-mat gap-mat clad-mat'

    power_fraction = '0.43116 0.0 0.0'
    power_shape_function = ppf_axial
  [../]
  [./CH4_LP]
    type = PBPipe
    eos = eos
    position = '0 0.5 -0.6'
    orientation = '0 0 1'

    A = 0.14769
    Dh = 2.972e-3
    length = 0.6
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./CH4_UP]
    type = PBPipe
    eos = eos
    position = '0 0.5 0.8'
    orientation = '0 0 1'
    A = 0.14769
    Dh = 2.972e-3
    length = 1.5
    n_elems = 2
    radius_i = 0.02

    lam_factor = 1.406
    turb_factor = 1.12933
    HTC_geometry_type = Pipe # pipe model

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.0005
    n_wall_elems = 1
    HT_surface_area_density_wall = 1107.8
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./Branch_CH4_L]
    type = PBSingleJunction
    inputs = 'CH4_LP(out)'
    outputs = 'CH4(in)'
    eos = eos
  [../]
  [./Branch_CH4_U]
    type = PBSingleJunction
    inputs = 'CH4(out)'
    outputs = 'CH4_UP(in)'
    eos = eos
  [../]


  [./CH5]
    type = PBCoreChannel
    eos = eos
    position = '0 1 0'
    orientation = '0 0 1'

    A = 0.153955129
    Dh = 1.694e-3
    length = 0.8
    n_elems = 4

    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density = 2113.6

    dim_hs = 1
    name_of_hs = 'fuel clad'
    Ts_init = 628.15
    n_heatstruct = 2
    fuel_type = cylinder
    width_of_hs = '6.32340e-3 7.0260e-4'
    elem_number_of_hs = '2 1'
    material_hs = 'fuel-mat clad-mat'

    power_fraction = '0.02860 0.0'
    power_shape_function = ppf_axial
  [../]
  [./CH5_LP]
    type = PBPipe
    eos = eos
    position = '0 1 -0.6'
    orientation = '0 0 1'
    A = 0.153955129
    Dh = 1.694e-3
    length = 0.6
    n_elems = 2
    radius_i = 0.02

    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density_wall = 2113.6

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.001 #0.0035
    n_wall_elems = 1
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./CH5_UP]
    type = PBPipe
    eos = eos
    position = '0 1 0.8'
    orientation = '0 0 1'

    A = 0.153955129
    Dh = 1.694e-3

    length = 1.5
    n_elems = 2
    radius_i = 0.02

    HTC_geometry_type = Pipe # pipe model
    HT_surface_area_density_wall = 2113.6

    dim_wall = 1
    Twall_init = 628.15
    wall_thickness = 0.001 #0.0035
    n_wall_elems = 1
    material_wall = ss-mat
    HS_BC_type = Adiabatic
  [../]
  [./Branch_CH5_L]
    type = PBSingleJunction
    inputs = 'CH5_LP(out)'
    outputs = 'CH5(in)'
    eos = eos
  [../]
  [./Branch_CH5_U]
    type = PBSingleJunction
    inputs = 'CH5(out)'
    outputs = 'CH5_UP(in)'
    eos = eos
  [../]

  [./IHX]
    type = PBHeatExchanger
    eos = eos
    eos_secondary = eos
    position = '0 1.5 5.88'
    orientation = '0 0 -1'
    A = 0.766
    A_secondary = 0.517
    Dh = 0.0186
    Dh_secondary = 0.014
    length = 3.71
    n_elems = 4
    initial_V_secondary = -2

    HTC_geometry_type = Pipe # pipe model
    HTC_geometry_type_secondary = Pipe
    HT_surface_area_density = 729
    HT_surface_area_density_secondary = 1080.1

    Twall_init = 628.15
    wall_thickness = 0.0033

    dim_wall = 1
    material_wall = ss-mat
    n_wall_elems = 1
  [../]

  [./pump_pipe]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 -1.5 3.61'
    orientation = '0 0 -1'

    A = 0.132
    Dh = 0.34
    length = 4.38
    n_elems = 4
    f = 0.001
    Hw = 0
  [../]

  [./pump_discharge]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 -1.5 -0.77'
    orientation = '0 1 0'

    A = 5.36
    Dh = 1
    length = 1.26
    n_elems = 2
    f = 0.001
    Hw = 0
  [../]


  [./inlet_plenum]
    type = PBVolumeBranch
    center = '0 0 -0.77'
    inputs = 'pump_discharge(out)'
    outputs = 'CH1_LP(in) CH2_LP(in) CH3_LP(in) CH4_LP(in) CH5_LP(in)'
    K = '0.2     0.5  5.2    6.0      13.8  12480'
    Area = 0.44934
    volume = 3.06

    initial_P = 3e5
    initial_T = 628.15
    eos = eos
    display_pps = true
    nodal_Tbc = true
  [../]

  [./hot_pool]
    type = PBLiquidVolume
    center = '0 0 6.45'
    inputs = 'CH1_UP(out) CH2_UP(out) CH3_UP(out) CH4_UP(out) CH5_UP(out)'
    outputs = 'IHX(primary_in)'
    K = '0.5 0.5 0.5 0.5 0.5 5.0'
    Area = 11.16
    volume = 92.51

    initial_level = 2.16 #3.59
    initial_T = 783.15
    initial_V = 0.00356
    display_pps = true
    eos = eos
    covergas_component = 'cover_gas'
  [../]

  [./cold_pool]
    type = PBLiquidVolume
    center = '0 0 2.3'
    inputs = 'IHX(primary_out) DHX(primary_out)'
    outputs = 'pump_pipe(in) DHX(primary_in)'

    K = '0.1 0.1 0.2  0.1 '
    Area =  23.96
    volume = 152.97

    initial_level = 5
    initial_T = 628.15
    initial_P = 3e5
    display_pps = true
    eos = eos
    covergas_component = 'cover_gas'
  [../]
  [./cover_gas]
    type = CoverGas
    n_liquidvolume =2
    name_of_liquidvolume = 'hot_pool cold_pool'
    initial_P = 1e5
    initial_Vol = 66.77
    initial_T = 783.15
  [../]
  [./Pump_p]
    type = PBPump
    eos = eos
    inputs = 'pump_pipe(out)'
    outputs = 'pump_discharge(in)'

    K = '1. 1.'
    Area = 0.055
    initial_P = 3e5

    Head = 415100
    Head_fn = pump_p_coastdown
  [../]

######  Secondary Loop  ######
  [./pipe8]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 2.7 2.17'
    orientation = '0 -1 0'

    A = 0.092
    Dh = 0.34
    length = 1
    n_elems = 2
    f = 0.001
    Hw = 0
  [../]

  [./pipe9]
    type = PBOneDFluidComponent
    eos = eos
    position = '0 1.7 5.88'
    orientation = '0 1 0'

    A = 0.092
    Dh = 0.34
    length = 1
    n_elems = 2
    f = 0.001
    Hw = 0
  [../]

  [./NaHX]
    type = PBHeatExchanger
    eos = eos
    eos_secondary = eos
    position = '0 2.7 5.88'
    orientation = '0 0 -1'
    A = 0.766
    A_secondary = 0.517
    Dh = 0.0186
    Dh_secondary = 0.014
    length = 3.71
    n_elems = 4
    initial_V_secondary = -2.8

    HTC_geometry_type = Pipe # pipe model
    HTC_geometry_type_secondary = Pipe
    HT_surface_area_density = 729
    HT_surface_area_density_secondary = 1080.1

    Twall_init = 628.15
    wall_thickness = 0.0008 #0.00174, 0.00087

    dim_wall = 1
    material_wall = ss-mat
    n_wall_elems = 1
  [../]

  [./Branch8]
    type = PBBranch
    inputs = 'pipe8(out)'
    outputs = 'IHX(secondary_in)'
    K = '0.05 0.05'
    Area = 0.092
    initial_P = 2e5
    eos = eos
  [../]

  [./Branch9]
    type = PBBranch
    inputs = 'IHX(secondary_out)'
    outputs = 'pipe9(in)'
    K = '0.0 0.0'
    Area = 0.092
    initial_P = 2e5
    eos = eos
  [../]
  [./Branch10]
    type = PBBranch
    inputs = 'pipe9(out) '
    outputs = 'NaHX(primary_in)'
    K = '0.01 0.01 '
    Area = 0.092
    initial_P = 2e5
    eos = eos
  [../]

  [./Pump_s]
    type = PBPump
    eos = eos
    inputs = 'NaHX(primary_out)'
    outputs = 'pipe8(in)'

    K = '0.1 0.1'
    Area = 0.766

    initial_P = 2e5

    Head = 40300
    Head_fn = pump_s_coastdown
  [../]

  [./secondary_p]
    type = ReferenceBoundary
    input = 'NaHX(primary_in)'
    variable = 'pressure'
    value = 1e5
  [../]

######  Power conversion loop  ######

  [./NaLoop_in]
    type = PBTDJ
    input = 'NaHX(secondary_in)'
    v_fn = flow_secondary
    T_bc = 596.75
    eos = eos
    weak_bc = true
  [../]

  [./NaLoop_out]
    type = PressureOutlet
    input = 'NaHX(secondary_out)'
    p_bc = '1e5'
    eos = eos
  [../]

######   DRACS loop    ######
  [./DHX]
    type = PBHeatExchanger
    eos = eos
    eos_secondary = eos
    position = '0 -1.5 6.04'
    orientation = '0 0 -1'
    A = 0.024
    A_secondary = 0.024
    Dh = 0.037
    Dh_secondary = 0.037
    length = 2.35
    n_elems = 4
    HTC_geometry_type = Pipe # pipe model
    HTC_geometry_type_secondary = Pipe
    HT_surface_area_density = 108.1
    HT_surface_area_density_secondary = 108.1

    Twall_init = 628.15
    dim_wall = 1
    wall_thickness = 0.0045
    material_wall = ss-mat

    n_wall_elems = 1
  [../]

  [./DRACS_inlet]
    type = PBTDJ
    input = 'DHX(secondary_in)'
    v_fn = flow_dhx
    T_bc = 450.3
    eos = eos
    wall_bc = true
  [../]
  [./DRACS_outlet]
    type = PressureOutlet
    input = 'DHX(secondary_out)'
    p_bc = 1.3e5
    eos = eos
  [../]
[]

[Postprocessors]
  [./pump_flow]
    type = ComponentBoundaryFlow
    input = pump_pipe(in)
  [../]
  [./IHX_primaryflow]
    type = ComponentBoundaryFlow
    input = IHX(primary_in)
  [../]
  [./IHX_secondaryflow]
    type = ComponentBoundaryFlow
    input = IHX(secondary_in)
  [../]
  [./DHX_flow]
    type = ComponentBoundaryFlow
    input = DHX(primary_in)
  [../]
  [./IHX_inlet_T]
    type = ComponentBoundaryVariableValue
    input = IHX(primary_in)
    variable = temperature
  [../]
  [./CH1_velocity]
    type = ComponentBoundaryVariableValue
    input = CH1(in)
    variable = velocity
  [../]
  [./CH2_velocity]
    type = ComponentBoundaryVariableValue
    input = CH2(in)
    variable = velocity
  [../]
  [./CH3_velocity]
    type = ComponentBoundaryVariableValue
    input = CH3(in)
    variable = velocity
  [../]
  [./CH4_velocity]
    type = ComponentBoundaryVariableValue
    input = CH4(in)
    variable = velocity
  [../]
  [./CH5_velocity]
    type = ComponentBoundaryVariableValue
    input = CH5(in)
    variable = velocity
  [../]
  [./CH1_outlet_flow]
    type = ComponentBoundaryFlow
    input = CH1_UP(out)
  [../]
  [./CH2_outlet_flow]
    type = ComponentBoundaryFlow
    input = CH2_UP(out)
  [../]
  [./CH3_outlet_flow]
    type = ComponentBoundaryFlow
    input = CH3_UP(out)
  [../]
  [./CH4_outlet_flow]
    type = ComponentBoundaryFlow
    input = CH4_UP(out)
  [../]
  [./CH5_outlet_flow]
    type = ComponentBoundaryFlow
    input = CH5_UP(out)
  [../]
  [./CH1_outlet_T]
    type = ComponentBoundaryVariableValue
    input = CH1_UP(out)
    variable = temperature
  [../]
  [./CH2_outlet_T]
    type = ComponentBoundaryVariableValue
    input = CH2_UP(out)
    variable = temperature
  [../]
  [./CH3_outlet_T]
    type = ComponentBoundaryVariableValue
    input = CH3_UP(out)
    variable = temperature
  [../]
  [./CH4_outlet_T]
    type = ComponentBoundaryVariableValue
    input = CH4_UP(out)
    variable = temperature
  [../]
  [./CH5_outlet_T]
    type = ComponentBoundaryVariableValue
    input = CH5_UP(out)
    variable = temperature
  [../]
  [./max_Tcoolant_core]
    type = NodalMaxValue
    block = 'CH1:pipe CH2:pipe CH3:pipe CH4:pipe'
    variable = temperature
  [../]
  [./max_Tco_core]
```

```
    type = NodalMaxValue
    block = 'CH1:pipe CH2:pipe CH3:pipe CH4:pipe'
    variable = Tw
  [../]
  [./max_Tci_core]
    type = NodalMaxValue
    block = 'CH1:solid:clad CH2:solid:clad CH3:solid:clad CH4:solid:clad'
    variable = T_solid
  [../]
  [./max_Tf_core]
    type = NodalMaxValue
    block = 'CH1:solid:fuel CH2:solid:fuel CH3:solid:fuel CH4:solid:fuel'
    variable = T_solid
  [../]
  [./max_Tcoolant_Ref]
    type = NodalMaxValue
    block = 'CH5:pipe'
    variable = temperature
  [../]
  [./max_Tco_Ref]
    type = NodalMaxValue
    block = 'CH5:pipe'
    variable = Tw
  [../]
  [./max_Tci_Ref]
    type = NodalMaxValue
    block = 'CH5:solid:clad'
    variable = T_solid
  [../]
  [./max_Tf_Ref]
    type = NodalMaxValue
    block = 'CH5:solid:fuel'
    variable = T_solid
  [../]

  [./DHX_heatremoval]
    type = HeatExchangerHeatRemovalRate
    block = 'DHX:primary_pipe'
    heated_perimeter = 2.5944
  [../]
  [./IHX_heatremoval]
    type = HeatExchangerHeatRemovalRate
    block = 'IHX:primary_pipe'
    heated_perimeter = 558.414
  [../]
  [./NaHX_heatremoval]
    type = HeatExchangerHeatRemovalRate
    block = 'NaHX:secondary_pipe'
    heated_perimeter = 558.414
  [../]
[]

[Preconditioning]
   active = 'SMP_PJFNK'
  [./SMP_PJFNK]
    type = SMP
    full = true
    solve_type  = 'PJFNK'
    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]
[] # End preconditioning block

[Executioner]
```

```
  type = Transient
  dt = 0.1
  dtmin = 1e-3

  # setting time step range
  [./TimeStepper]
    type = FunctionDT
    time_t = '-1000   -499.9    -499.8    -499    -498    -450   -449    -1      0        2      3
       10    11   380    381     440    441    1e5'
    time_dt =' 0.02     0.02       0.2     0.2     0.5     0.5      2     2    0.2      0.2    0.5
      0.5     2     2      2       2      2      2'
    min_dt = 1e-3
  [../]

  nl_rel_tol = 1e-7
  nl_abs_tol = 1e-6
  nl_max_its = 10

  l_tol = 1e-4
  l_max_its = 100
  line_search = basic

  start_time = -500
  num_steps = 10000
  end_time = 0

  [./Quadrature]
    type = SIMPSON
    order = SECOND
  [../]
[] # close Executioner section

[Outputs]
  print_linear_residuals = false
  [./out_displaced]
    type = Exodus
    use_displaced = true
    execute_on = 'initial timestep_end'
    sequence = false
  [../]
  [./checkpoint]
    type = Checkpoint
    num_files = 1
  [../]
  [./console]
    type = Console
    perf_log = true
  [../]
  [./csv]
    type = CSV
  []
[]
```

5.8 Uncertainty Quantification using Dakota

An example problem demonstrating a coupling framework with Dakota for performing uncertainty quantification study is included here. This example covers two modes of execution (on a single ’local’ node or on multiple nodes) and is based on the Single Channel Flow problem covered in section 5.2. For more information regarding this example, please refer to chapter 6.

6 Uncertainty Quantification

This section demonstrates an integration between SAM and the Dakota statistical analysis toolkit developed by Sandia National Laboratory [19]. The integration of the two codes is achieved through a Python coupling framework in order to leverage the sensitivity analysis, uncertainty quantification, and many other statistical tools available through Dakota. The coupling framework is provided through two examples that are available in the Examples directory of the SAM package.

6.1 Acquiring and Installing Dakota

Dakota is publicly available at no cost and is distributed by Sandia National Laboratory. Dakota is supported on most operating systems, including Unix, Linux, and Windows. Users should download the binary executable for preferred platform or compile Dakota from source code where needed. Please refer to the Dakota website for installation instructions: `https://dakota.sandia.gov/quickstart.html`

6.2 Dakota and SAM Coupling

At the base level, Dakota interfaces with the code in a black-box fashion, using system calls to initiate simulations with SAM. Data communication between Dakota and the external code occurs through parameter and response files. The coupling searches a SAM template input file and replaces the uncertain parameters with the random values generated by Dakota. Once all SAM inputs are generated, execution can be handled either on a serial node (with one or more cores on a single machine) through Dakota's built-in management schemes, or they can be handled on multiple nodes by batching and submission to a cluster queue to be handled by a scheduler on an HPC (see Section 6.3).

After the SAM simulations complete, the Python interface reads the output CSV file and searches for the target responses. The coupling driver does not currently support Exodus file output, so it is the user's responsibility to ensure that response parameters (PostProcessors) are printed through the MOOSE CSV output file options. The built-in filters can process the SAM simulation results before sending to Dakota: `max` for the maximum value, `min` for the minimum value, `begin` for the beginning value of the simulation, and `end` for the ending value of the simulation. Users can choose the parameters or response variables on which these filters are applied. The responses of interest are written in a result file and sent back to Dakota for statistics. This is summarized in Figure 6.1.

6.3 Multiple Node Execution (HPC cluster)

Although Dakota has built-in methods to utilize a multiple node setup (i.e. high-performance cluster) for the execution step, Dakota is unable to exercise this option if the underlying coupled code also utilizes MPI. As an alternative, a framework has been implemented to decouple the Dakota pre-processing and post-processing steps from actual code execution, allowing for greater flexibility. Overall, the procedure follows:

1. **Pre-processing**: Dakota parses the defined input parameters and generates requested coupled code input files into separate work directories. A random number generator seed is used to maintain consistency. Dakota also generates "dummy" results in order to close out the pre-processing run.

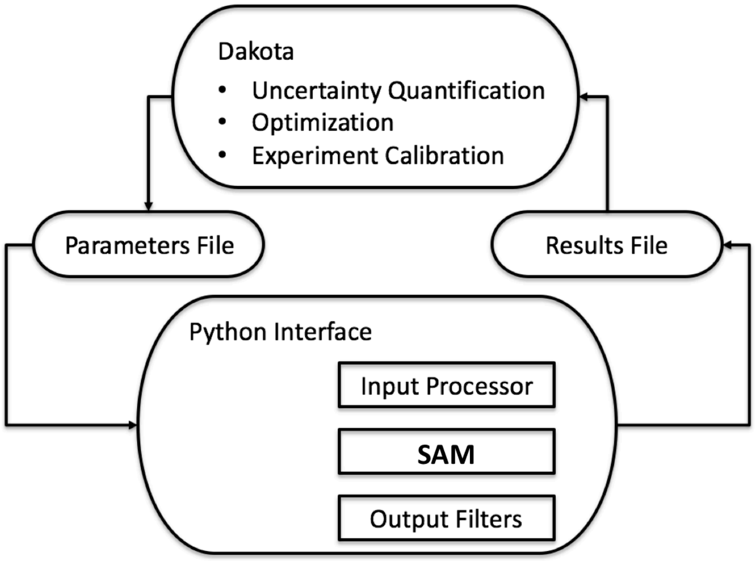


Figure 6.1: Dakota-SAM Coupling Scheme

2. **Execution**: All code input files are staged for submission to an HPC cluster job queue. Depending on the simulation requirements, it is suggested to batch multiple jobs together onto a computational node to maximize utilization of available resources. An example script is provided in `DakotaBatch.py`.

3. **Post-processing**: After all executions are finished, the user executes a Dakota post-processing step to pull the generated results and obtain the desired statistics and information. The random number generator seed maintains consistency, as the input parameters are re-generated in this step and are needed for statistical correlation.

6.4 File structure overview

This section will briefly outline the different files required for a statistical study performed using this Dakota-SAM coupled framework. These files are included in the UQ examples included in the SAM examples directory.

- `DakotaCouplingLib.py`

  This file contains several Python classes that allows Dakota to interact with the SAM input files and perform code execution. It also includes post-processing tools to interpret SAM output. This file should not be modified and should be kept in an accessible location.

- `Dakota Code Driver File`

  This file interfaces with `DakotaCouplingLib.py` to define the Dakota model information needed for a particular study.

  – `DakotaCodeDriver.py`

  For a standard study performed on a single node, this driver file will handle pre-processing, execution, and post-processing of results. This file requires paths to the code executable and coupling library, as well as specifications of your output variables.

  - `DakotaPreDriver.py`

    For a study performed on multiple nodes, this driver file will only handle pre-processing of the problem. This file requires the path to the coupling library and specification of the output.

  - `DakotaPostDriver.py`

    For a study performed on multiple nodes, this driver file will only handle post-processing. This file requires the path to the coupling library and specification of the code and the output.

- `dakota_sam.in`

  This is the Dakota input file that defines the statistical study being performed. Please refer to the example in Section 6.6.1 or the Dakota user manual [19] for more detail.

- `ex01.template`

  This is the SAM input file template where perturbed parameters are replaced with {wildcard} placeholders corresponding to the Dakota input file. Note that the default file extension that the coupling library searches for is `*.template`, other file extensions must be explicitly claimed in the Dakota code driver file and the Dakota input file. {wildcard} placeholders can be specified as follows:

  - `{parameter_name}`: parameter value replacement
  - `{parameter_name | format}`: parameter value replacement with formatting specification (e.g. 12.5E)
  - `{parameter_name * 0.02}`: arithmetic operations can be performed on the sampled parameter value

- `DakotaBatch.py`

  This is a Python script that can be used in a multiple node execution job to group together processed input files into "batched submissions" to maximize available computational resources. This file requires specification of the number of processors per node (or input files to be batched to each node), the name of the input file, and the command to execute SAM.

- `pbs-uq.header` under `queuedir`

  This is a template PBS submission script header for the generated batched submission jobs written by `DakotaBatch.py`. Modify or substitute this header as needed to suit each HPC's requirements.

- `run.sh`

  There are several shell scripts included to automate the Dakota job process, users are encouraged to use or reference these scripts.

6.5 Example case: single node execution

Here we demonstrate UQ coupling with SAM using Latin hypercube sampling to sample input parameters in a Single Channel Flow problem (see 5.2). This example covers the case of executing your Dakota study on a single computational node. A standard 1-D fluid channel is given a defined inlet temperature and fluid velocity and outlet pressure boundary condition. The peak outlet temperature is taken as the response variable in this example. Sodium is the working fluid in this model.

The variables perturbed in this example include coverage for geometry (typical parameters), closure model parameters, and equation-of-state parameters. Table 6.1 indicates the variables perturbed along with their nominal value and uncertainty range.

Table 6.1: Nominal Values of Uncertain Parameters

Uncertain Parameters	Nominal	Uncertainty
Hydraulic Diameter	1.0 m	1% (uniform)
Pipe Flow Heat Transfer Coefficient	1.0	50% (normal)
Sodium Thermal Conductivity	1.0	15% (normal)

6.5.1 Dakota Input File

The Dakota input file (i.e. `dakota_sam.in`) defines the method, variables, interface, and responses of interest.

The sampling method is identified in the method section (keyword: `random`).

```
method,
        sampling
                sample_type random
                samples = 8
                probability_levels =    0.05 0.95
```

Uncertain variables, probability distributions, and upper/lower bounds are specified in the variable section.

```
variables,
        uniform_uncertain = 1
                lower_bounds            0.99
                upper_bounds            1.01
                descriptors             'hyd_diam'

        normal_uncertain = 2
                means           1.0                     1.0
                std_deviations  0.5                     0.15
                descriptors     'system_htc'            'Na_k'
```

The interface section defines the driver file name, the parameter file saving the random values, SAM template input file, and the response file for SAM simulation results. Note that the code input template format is specified in the `copy_files` parameter. The number of concurrent SAM simulations is specified by the `asynchronus_evaluation_concurrency` parameter, typically limited by the number of processors available to the local node. The `parameters_file` and `results_file` should be specified as below, which correspond to the default filenames used by the Python coupling interface. Any files that need to be copied to each work directory should be specified by the copy_files parameter. Here, the SAM template file (with file extension *.template) is designated for copying.

```
interface,
        fork
                asynchronous_evaluation_concurrency = 4
                analysis_driver = 'DakotaCodeDriver.py'
                parameters_file = 'params.in'
                results_file    = 'results.out'
                work_directory directory_tag
                copy_files = '*.template'
                named 'workdir' file_save  directory_save
                aprepro
                deactivate active_set_vector
```

The total number of the responses is claimed as response_functions, and the target responses along with the criteria are specified in the driver file (i.e. DakotaCodeDriver.py).

```
responses,
        response_functions = 2
        descriptors 'outlet_temperature' 'outlet_flow'
        no_gradients
        no_hessians
```

6.5.2 Dakota-SAM Coupling File

This example covers the case of Single Node Execution without multiple processors or parallel instances of execution. In this scenario, the entire process is handled in one step by Dakota using the Python driver files. Dakota is initiated to drive the SAM simulations via a driver file (i.e. DakotaCodeDriver.py). The commands below load the coupling libraries along with the SAM executable files. Appropriate paths must be specified for the SAM executable (CodeExecutableDirectory) and the DakotaCouplingLibrary.py file (CouplingLibraryDirectory), either with full paths or relative to the work directories.

```
Code = 'SAM'
CodeExecutableDirectory = '/path/to/sam/'
CouplingLibraryDirectory = '../../'
sys.path.append(CouplingLibraryDirectory)
from DakotaCouplingLib import DakotaCouplingInterface
UQInterface = DakotaCouplingInterface()
```

The specified output is also indicated here in the driver file.

```
TrackedCodeOutput = {'outlet_temperature':['outlet_temp','max']},
    'outlet_flow':['outlet_flow','min']}
```

Note the format for each output response should be specified as:

```
{'dakota_response_name':['code_output_variable_name','statistic_method']}
```

Lastly, the coupling library interface will search for the SAM input template which was copied into each work directory by Dakota, as specified in the Dakota input file. The interface will identify the template if named using the default file extension *.template. If the template file follows a different naming convention, it can be explicitly claimed in the code driver using:

```
TemplateFilemame = 'ex01.template'
```

The interface searches the template for the wildcards specifying the input parameters, marked by brackets. Please refer to section 6.4 for more information regarding the template file.

The coupled simulation parameters as specified are passed to the interface:

```
UQInterface.RunInterfaceFunctions(Code,sys.argv[1],CodeExecutableDirectory,
  sys.argv[2],TrackedCodeOutput)

# If the template filename is declared:
UQInterface.RunInterfaceFunctions(Code,sys.argv[1],CodeExecutableDirectory,
  sys.argv[2],TrackedCodeOutput,TemplateFilename)
```

Finally, the following command is executed to run Dakota, which can be found in the `run_pre.sh` script that also prepends several commands to clean the directory of unnecessary files.

```
dakota -i dakota_sam.in -o dakota_sam.out
```

6.5.3 Dakota Output

Dakota output records every evaluation and is organized into three parts: problem information, parameters and responses, and summary statistics and can be found in the Dakota output file (`dakota_sam.out`). For the uncertainty quantification analysis, Dakota generates random values based on user-specified ranges and distributions. Parameters and responses for each evaluation are included in the output file as well as summarized in the `Dakota_Coupling_Summary.csv` file.

Means, standard deviations, and 95% confidence intervals are computed for each response. In addition, Dakota calculates several statistics between uncertainties and responses of interest, such as covariance, Pearson coefficient, simple, partial, and rank correlations. Please refer to the Dakota User's Manual [19] for more information on output and analysis options.

6.6 Example case: multiple node execution

Here we demonstrate UQ coupling with SAM using LHS to sample input parameters in a trivial pipe flow case as previously defined (see 6.5). This example covers the case of parallel execution, such as on a HPC cluster or multi-processor resource, demonstrating the workflow as described in Section 6.3. The following will discuss the necessary modifications to the single node execution example to split the process into pre-processing, execution, and post-processing steps.

6.6.1 Dakota Input File

The Dakota input file defines the method, variables, interface, and responses of interest. There are two separate files for the pre-processing (i.e. `dakota_sam_pre.in`) and post-processing (i.e. `dakota_sam_post.in`) steps. These files are almost identical, but use a matching random number generator seed for consistent sampling of the input parameter space.

The sampling method is identified in the method section (keyword: `random`) with `seed` specified.

```
method,
sampling
        sample_type random
        seed = 61820
        samples = 16
        probability_levels =    0.05 0.95
```

The interface section defines the driver file name which differ for the pre- and post-processing step, specified by analysis_driver. For clarity, the example case includes two separate Dakota input files for the pre- and post-processing steps, but in practice the only difference between the two files is the specification of the analysis_driver parameter to the correct Python code driver file.

```
interface,
        fork
            analysis_driver = 'DakotaPreDriver.py'
            parameters_file = 'params.in'
            results_file    = 'results.out'
            work_directory directory_tag
            copy_files = '*.template'
            named 'workdir'  file_save  directory_save
            aprepro
            deactivate active_set_vector
```

6.6.2 Dakota-SAM Coupling File

For parallel execution on multiple processors, the workflow is divided into a pre- and post-processing step so that the MPI processes do not conflict. There are two similar driver files that differ on the final library command to interface the input specifications with the Dakota UQ interface.

The pre-processor driver in DakotaPreProcess.py passes the library command:

```
UQInterface.PreProcess(sys.argv[1],sys.argv[2],TrackedCodeOutput)
```

The pre-processor driver in DakotaPostProcess.py passes the library command:

```
UQInterface.PostProcess(Code,sys.argv[1],sys.argv[2],TrackedCodeOutput)
```

To run the pre-processor, use the following command to run Dakota, or use the example run_pre.sh batch file to clean the work directory and run Dakota.

```
dakota -i dakota_sam_pre.in -o dakota_sam_pre.out
```

After running the pre-processor, the work directories and sampled input files should have been generated and made ready for parallel execution. For HPC clusters with a queue system, it is suggested that input files be grouped ("batched") together to match the number of input files to the number of processors per node. An example of such script for batching is provided in DakotaBatch.py, which uses a template PBS header for job submissions. After execution, each directory should contain its output files ready for post.

To run the post-processor, use the following command to run Dakota, or use the example run_post.sh script. For large runs, Dakota can be executed in parallel using mpirun.

```
dakota -i dakota_sam_post.in -o dakota_sam_post.out
```

6.6.3 Dakota Output

The format of the Dakota output is the same regardless of execution method, please refer to Section 6.5.3 or the Dakota User's Manual [19] for more information.

ACKNOWLEDGMENTS

The SAM code development is funded by U.S. Department of Energy Office Nuclear Energy Advanced Modeling and Simulation (NEAMS) program, with additional supports from multiple sources including Argonne Laboratory Directed Research and Development (LDRD), DOE-NE Industry Funding Opportunity Announcement (iFOA), Gateway for Accelerated Innovation in Nuclear (GAIN), Technology Commercialization Fund (TCF), and the U.S. Nuclear Regulatory Commission.

REFERENCES

[1] R. Hu. SAM Theory Manual. Technical Report ANL/NE-17/4, Argonne National Laboratory, 2017.

[2] R. Hu. Three-dimensional flow model development for thermal mixing and stratification modeling in reactor system transients analyses. *Nuclear Engineering and Design*, 345:209–215, 2019.

[3] G. Hu, G. Zhang, and R. Hu. Reactivity Feedback Modeling in SAM. Technical Report ANL-NSE-19/1, Argonne National Laboratory, 2019.

[4] G. Hu, R. Hu, and L. Zou. Development of Heat Pipe Reactor Modeling in SAM. Technical Report ANL-NSE-19/9, Argonne National Laboratory, 2019.

[5] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandié. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design*, 239:1768–1778, 2009.

[6] B.S. Kirk, J.W. Peterson, R.H. Stogner, and F.C. Graham. libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22:237–254, 2006.

[7] PETSc. PETSc Web page., 2017. URL http://www.mcs.anl.gov/petsc.

[8] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov Methods: a Survey of Approaches and Applications. *Journal of Computational Physics*, 193:357–397, 2004.

[9] R.A. Berry, J.W. Peterson, H. Zhang, R.C. Martineau, H. Zhao, L. Zou, D. Andrs, and J. Hansel. RELAP-7 Theory Manual. Technical Report INL/EXT-14-31366/Revision 3, Idaho National Laboratory, 2018.

[10] R. Hu, J.W. Thomas, E. Munkhzul, and T.H. Fanning. Coupled System and CFD Code Simulation of Thermal Stratification in SFR Protected Loss-Of-Flow Transients. In *Proceedings of ICAPP 2014*, 2014.

[11] T.H. Fanning and R. Hu. Coupling the System Analysis Module with SAS4A/SASSYS-1. Technical Report ANL/NE-16/22, Argonne National Laboratory, 2016.

[12] R. Martineau, D. Andrs, J. Hansel, C. Permann, M. Bernard, R. Johns, R. Hu, J. Wolf, H. Zhang, and R. Szilard. Extending the Capability of Nuclear Plant Systems Analysis with Advanced Tightly-Coupled Nuclear Fuels Performance. In *2018 American Nuclear Society Annual Meeting*, 2018.

[13] HYPRE Web page. URL https://computation-rnd.llnl.gov/linear_solvers/.

[14] MPICH2 Web page. URL http://www.mcs.anl.gov/mpich2.

[15] TBB Web page. URL https://www.threadingbuildingblocks.org/.

[16] The RELAP5-3D Code Development Team. RELAP5-3D Code Manual Volume IV: Models and Correlations. Technical Report INL-EXT-98-00834, Revision 4, Idaho National Laboratory, 2014.

[17] U.S. NRC. TRACE 5.0 Assessment Manual-Appendix A: Fundamental Validation Cases. Technical report, U. S. Nuclear Regulatory Commission, 2008.

[18] Y.I. Chang, P.J. Finck, and C. Grandy. Advanced Burner Test Reactor Preconceptual Design Report. Technical Report ANL-ABR-1 (ANL-AFCI-173), Argonne National Laboratory, 2006.

[19] B.M. Adams, W.J. Bohnhoff, K.R. Dalbey, M.S. Ebeida, J.P. Eddy, M.S. Eldred, G. Geraci, R.W. Hooper, P.D. Hough, K.T. Hu, J.D. Jakeman, M. Khalil, K.A. Maupin, J.A. Monschke, E.M. Ridgway, A.A. Rushdi, J.A. Stephens, L.P. Swiler, D.M. Vigil, T.M. Wildey, and J.G. Winokur. Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.11 User's Manual. Technical Report SAND2014-4633, Sandia National Laboratory, 2019.